Code Jam 2021 - Round 2

Minimum Sort

Problem

In this problem, you need to sort a list of $\mathbf{N} = 100$ distinct integers in strictly increasing order. You can rearrange the list by swapping the contents of any two positions (they do not need to be adjacent). Unfortunately, you cannot read those contents directly. You can access information about the list contents by querying the minimum of a range. The minimum query gives you the *position* of the minimum value over a range of consecutive positions. For example, in the list $[51, 33, 100, 11]$, the minimum over the range between positions 2 and 4, inclusive (1-based), is at position 4 and the minimum between positions 1 and 3 is at position 2.

Queries about the minimum within a range are limited by a coin budget per test case. Larger ranges are cheaper: asking about the position of the minimum between positions i and j (for $i < j$) costs $\lceil 10^8/(j - i + 1) \rceil$ coins, where $\lceil x \rceil$ is the smallest integer greater than or equal to x (that is, x rounded up). Swap operations, on the other hand, do not cost any coins.

Write a program that sorts lists of integers using any number of swaps and at most 6×10^8 coins per test case distributed among any number of minimum queries.

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially, the judge will send you a single line containing two integers $\mathbf{T}$ and $\mathbf{N}$: the number of test cases and the number of elements to sort within each test case, respectively. The judge has the initial lists preset before it gets any input from your program, and the only changes done to them during the exchanges with your program are the swaps that you request.

Then, you must process $\mathbf{T}$ test cases. Each test case consists of a series of exchanges plus an additional line indicating you are done. Each exchange consists of you printing one line and the judge printing one line in response. Your program must print a single line containing one of these options:

- An uppercase `M` and two integers i and j with $i < j$ representing a minimum query. The judge responds with a single integer representing the position of the minimum value in the list within 1-based positions i and j, inclusive.
- An uppercase `S` and two integers i and j with $i < j$ representing a swap operation. The judge swaps the two elements at 1-based positions i and j and responds with `1`.
- An uppercase `D` representing that you are done sorting the list. The judge checks the list. It responds with `1` if the list is sorted in strictly increasing order and `-1` if it is not.

After the judge responds `1` to a `D`, it will finish if it was the last test case or it will immediately start waiting for your first command for the next test case. After receiving the judge's response for the $\mathbf{T}$-th case, your program must finish in order to not receive a Time Limit Exceeded error.

If the judge receives an invalidly formatted line or invalid values from your program at any moment, including a minimum operation whose cost would exceed your remaining budget for the test case, the judge will print a single number `-1`. After the judge prints `-1` for any of the reasons explained above, it will not print any further output. If your program continues to wait for the judge after receiving a `-1`, your program will time out, resulting in a Time Limit Exceeded

error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

Limits

Time limit: 60 seconds.
Memory limit: 1 GB.

Test Set 1 (Visible Verdict)

$\mathbf{T} = 100$.
$\mathbf{N} = 100$.

Sample Interaction

Judge **Solution**

Constraints

`2 4`
Judge provides $\mathbf{T}$, $\mathbf{N}$

Case 1
List: $[51, 33, 100, 11]$

`M 2 4`
Solution queries for the minimum in the range $[2, 4]$
This operation cost $\lceil 10^8/3 \rceil = 33333334$ *coins*

`4`
Judge provides the position of the minimum in the range

`M 1 3`
Solution queries for the minimum in the range $[1, 3]$
This operation cost $\lceil 10^8/3 \rceil = 33333334$ *coins*

`2`
Judge provides the position of the minimum in the range

`S 1 4`
Solution asks to swap the elements at positions 1 and 4

`1`
Judge responds 1 to a swap operation

The list is now $[11, 33, 100, 51]$

`M 3 4`
Solution queries for the minimum in the range $[3, 4]$
This operation cost $\lceil 10^8/2 \rceil = 50000000$ *coins*

`4`
Judge provides the position of the minimum in the range

`S 3 4`

Solution asks to swap the elements at positions 3 and 4

```
1
```

Judge responds 1 to a swap operation

The list is now $[11, 33, 51, 100]$

```
D
```

Solution tells the judge it is done sorting
The solution spent a total of 116666668 *coins*

```
1
```

Judge responds with 1 saying the list is properly sorted

Case 2
List: $[30, 20, 10, 40]$

```
M 1 4
```

Solution queries for the minimum in the range $[1, 4]$
This operation cost $\lceil 10^8/4 \rceil = 25000000$ *coins*

```
3
```

Judge provides the position of the minimum in the range

```
S 1 3
```

Solution asks to swap the elements at positions 1 and 3

```
1
```

Judge responds 1 to a swap operation

The list is now $[10, 20, 30, 40]$

```
M 3 4
```

Solution queries for the minimum in the range $[3, 4]$
This operation cost $\lceil 10^8/2 \rceil = 50000000$ *coins*

```
3
```

Judge provides the position of the minimum in the range

```
M 2 4
```

Solution queries for the minimum in the range $[2, 4]$
This operation cost $\lceil 10^8/3 \rceil = 33333334$ *coins*

```
2
```

Judge provides the position of the minimum in the range

```
D
```

Solution tells the judge it is done sorting
The solution spent a total of 108333334 *coins*

```
1
```

Judge responds with 1 saying the list is properly sorted

Both Cases Finished

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Code Jam 2021 - Round 2

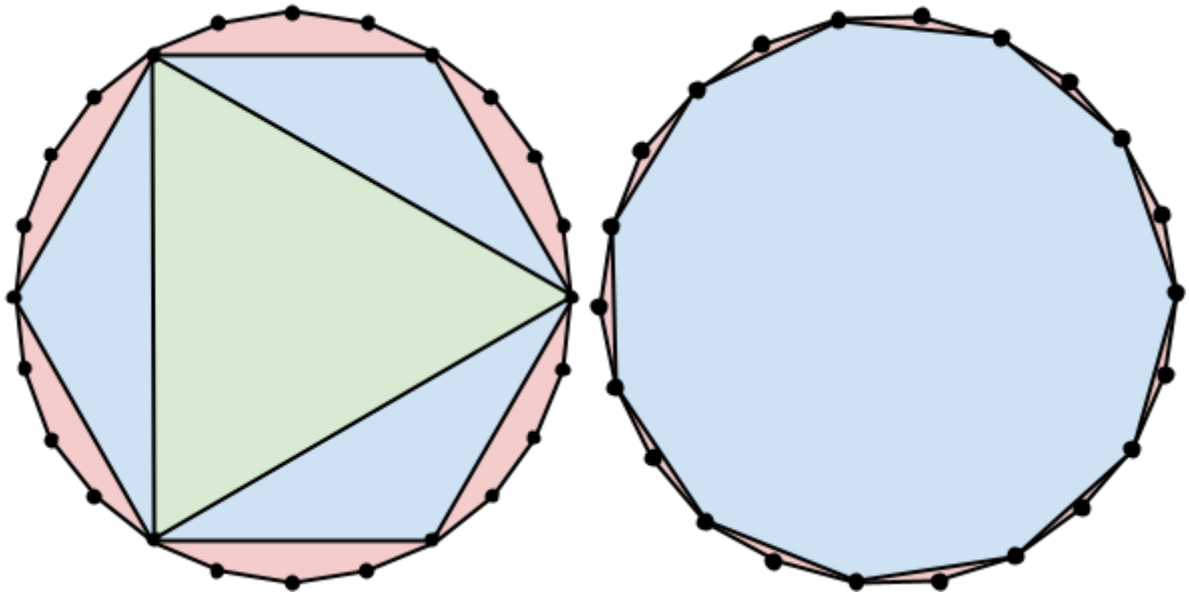
Matrygons

Problem

A matryoshka is a type of doll that originated in Russia over a century ago. Their defining characteristic is that they consist of a set of dolls, all of a different size, with smaller dolls fitting nicely inside larger dolls.

In this problem, we work with matrygons, which are sets of regular convex polygons that follow a similar nesting pattern. A matrygon consists of a set of regular convex polygons with positive area $p_1, p_2, \ldots, p_k$ such that, for all i, the vertices of p_{i+1} overlap with a proper subset of the vertices of p_i (p_{i+1} has strictly less vertices than p_i).

For example, the following pictures illustrates two matrygons. The first one contains 3 regular convex polygons: a regular icositetragon (24 sides), a regular hexagon (6 sides), and an equilateral triangle (3 sides). The second one contains 2 regular convex polygons: a regular icosidigon (22 sides) and a regular hendecagon (11 sides). Each of these matrygons has 33 total sides among all polygons in it.



Given a fixed total number of sides $\mathbf{N}$, calculate the largest number of polygons that can be part of a matrygon such that the total number of sides among all polygons in it is exactly $\mathbf{N}$.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ lines follow. Each line represents a test case and contains a single integer $\mathbf{N}$, the target total number of sides.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is the maximum number of polygons in a matrygon such that the total number of sides among all polygons in it is exactly $\mathbf{N}$.

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.

Test Set 1 (Visible Verdict)

Time limit: 20 seconds.
$3 \leq \mathbf{N} \leq 1000$.

Test Set 2 (Visible Verdict)

Time limit: 40 seconds.
$3 \leq \mathbf{N} \leq 10^6$.

Sample

Sample Input	Sample Output
3 33 15 41	Case #1: 3 Case #2: 2 Case #3: 1

The first matrygon pictured in the problem statement is an optimal solution for Sample Case #1.

In Sample Case #2, we can get to two polygons by fitting a regular pentagon (5 sides) inside a regular decagon (10 sides).

In Sample Case #3, there is no way to create a matrygon with multiple regular polygons, so our only option is to use a single regular tetracontahenagon (41 sides).

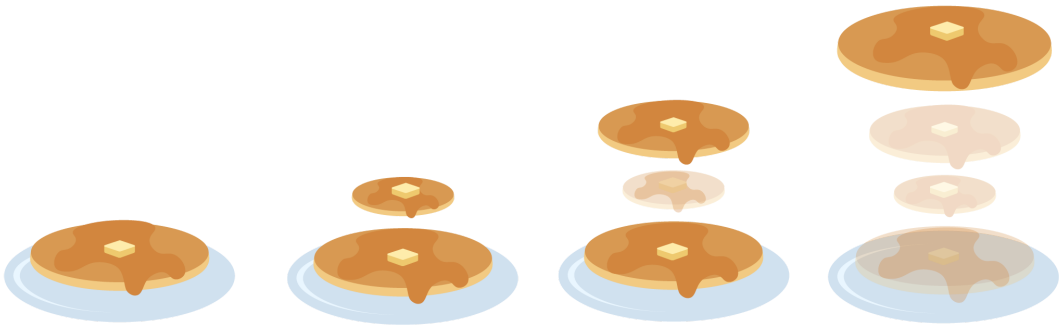
Code Jam 2021 - Round 2

Hidden Pancakes

Problem

We are cooking $\mathbf{N}$ pancakes in total. We cook one pancake with a 1 centimeter (cm) radius, one with a 2 cm radius, one with a 3 cm radius, ..., and one with an $\mathbf{N}$ cm radius, not necessarily in that order. After we cook the first pancake, we just lay it on a plate. After we cook each subsequent pancake, we lay it on top of the previously made pancake, with their centers coinciding. In this way, a pancake is visible from the top of the stack when we first add it. A pancake only becomes hidden if we later cook another pancake with a larger radius.

For example, say we cook 4 pancakes. We first cook the pancake with radius 3 cm, and it is visible. Then, we cook the pancake with radius 1 cm, lay it on top of the first one and both are visible. Third, we cook the pancake with radius 2 cm, and now that covers the previous pancake, but not the first one, so 2 pancakes remain visible in total. Finally, we cook the pancake with radius 4 cm which covers the other pancakes leaving only 1 visible pancake. The picture below illustrates the state of the stack after each pancake is cooked. Within each stack, the fully colored pancakes are visible and the semi-transparent pancakes are not visible.



Let $\mathbf{V_i}$ be the number of visible pancakes when the stack contains exactly i pancakes. In the example above, $\mathbf{V_1} = 1$, $\mathbf{V_2} = 2$, $\mathbf{V_3} = 2$, and $\mathbf{V_4} = 1$.

Given the list $\mathbf{V_1}, \mathbf{V_2}, \ldots, \mathbf{V_N}$, how many of the $\mathbf{N}!$ possible cooking orders yield those values? Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime $10^9 + 7$ (1000000007).

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow, each described with two lines. The first line of a test case contains a single integer $\mathbf{N}$, the number of pancakes we cook. The second line of a test case contains $\mathbf{N}$ integers $\mathbf{V_1}$, $\mathbf{V_2}$, ..., $\mathbf{V_N}$, representing the number of visible pancakes after we cook 1, 2, ..., $\mathbf{N}$ pancakes, respectively.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is the number of cooking orders of $\mathbf{N}$ pancakes that yield the given numbers of visible pancakes after each step, modulo the prime $10^9 + 7$ (1000000007).

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{V_i} \leq i$, for all i.

Test Set 1 (Visible Verdict)

Time limit: 30 seconds.
$2 \leq \mathbf{N} \leq 13$.

Test Set 2 (Hidden Verdict)

Time limit: 40 seconds.
$2 \leq \mathbf{N} \leq 10^5$.

Sample

Note: there are additional samples that are not run on submissions down below.

Sample Input

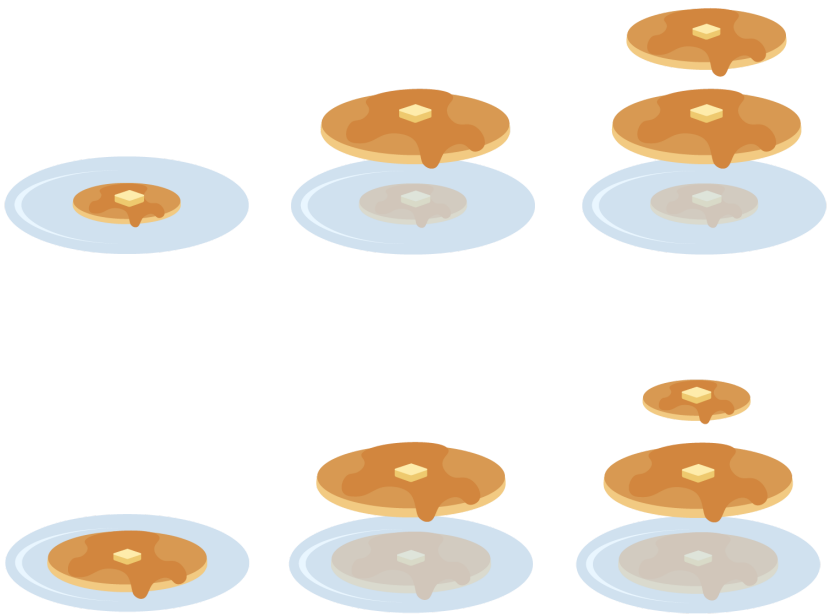
```
3
4
1 2 2 1
3
1 1 2
3
1 1 3
```

Sample Output

```
Case #1: 1
Case #2: 2
Case #3: 0
```

Sample Case #1 is explained in the problem statement. The order $3, 1, 2, 4$ is the only one that yields the given $\mathbf{V_i}$s.

In Sample Case #2, both the order $1, 3, 2$ and the order $2, 3, 1$ yield the intended $\mathbf{V_i}$s. The pictures below illustrate both options.



In Sample Case #3, only 1 pancake is visible after the second is made, so there is no way to have more than 2 visible pancakes by only adding a third.

Additional Sample - Test Set 2

The following additional sample fits the limits of Test Set 2. It will not be run against your submitted solutions.

Sample Input

```
1
24
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2
```

Sample Output

```
Case #1: 234141013
```

In the Sample Case for Test Set 2, there are 316234143225 cooking orders that yield the given $\mathbf{V_i}$s. Modulo $10^9 + 7$, this value is 234141013.

Code Jam 2021 - Round 2

Retiling

Problem

Cody-Jamal's latest artistic installment is a tiled kitchen floor that can be retiled to different patterns. The floor consists of a matrix of $\mathbf{R}$ rows and $\mathbf{C}$ columns of square tiles. Each tile is reversible, one side is magenta and the other one is green.

To retile the kitchen, there are two allowed operations:

- flip a tile, changing its visible color from magenta to green, or vice versa, and
- swap two adjacent tiles (horizontally or vertically, but not diagonally), without flipping either.

Viewing Cody-Jamal's artistic floor is free, but interacting with it is not. Performing a single flip operation costs $\mathbf{F}$ coins, and performing a single swap operation costs $\mathbf{S}$ coins.

You can see the current state of the floor and want to turn it into a particular pattern. What is the minimum amount of coins you need to spend to achieve your goal?

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. The first line of a test case contains 4 integers: $\mathbf{R}$, $\mathbf{C}$, $\mathbf{F}$ and $\mathbf{S}$, the number of rows and columns of the floor, the cost in coins of flipping and the cost in coins of swapping, respectively. Then, $2 \cdot \mathbf{R}$ lines follow. The first $\mathbf{R}$ lines contain $\mathbf{C}$ characters each. The j-th character of the i-th of these lines represents the current state of the tile in the i-th row and j-th column. The character is `M` if the currently visible side is magenta and `G` otherwise. The last $\mathbf{R}$ lines also contain $\mathbf{C}$ characters each. The j-th character of the i-th of these lines represents the color you want for the tile in the i-th row and j-th column, using the same character code as for the current state.

Output

For each test case, output one line containing `Case #`x`:` y, where x is the test case number (starting from 1) and y is the minimum amount of coins you need to spend to perform operations that allow you to change the tile colors from their current state to your intended one.

Limits

Time limit: 40 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{R} \leq 10$.
$1 \leq \mathbf{C} \leq 10$.

Test Set 1 (Visible Verdict)

$\mathbf{F} = 1$.
$\mathbf{S} = 1$.

Test Set 2 (Hidden Verdict)

$1 \leq \mathbf{F} \leq 10^6$.
$1 \leq \mathbf{S} \leq 10^6$.

Sample

Note: there are additional samples that are not run on submissions down below.

Sample Input

```
2
2 4 1 1
MGMG
MMMG
GMGM
MMMM
3 3 1 1
MGG
GMG
MMM
MMM
MGM
MMG
```

Sample Output

```
Case #1: 3
Case #2: 4
```

In Sample Case #1, there are 5 tiles that have a different color between the current and the desired states of the floor. Since each operation can change at most 2 tiles, at least 3 operations, costing 3 coins, are needed. One way to do it with exactly 3 coins is:

1. Swap the leftmost two tiles in the top row.
2. Swap the rightmost two tiles in the top row.
3. Flip the bottom right corner tile.

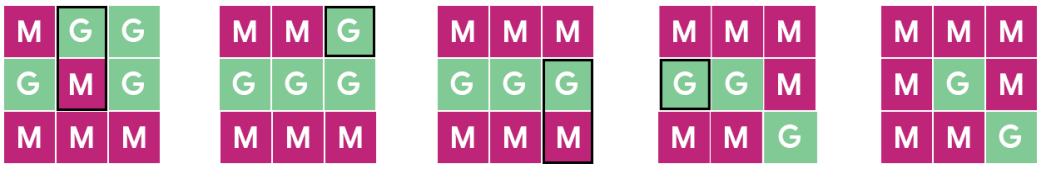
The picture below illustrates the states the floor goes through. The highlighted tile or tiles in each state are the ones being changed by the operation.



In Sample Case #2, there are 6 tiles that need changing. However, since only swaps can change two tiles at a time, solving it with 3 operations would require all of them to be swaps. There is no way to involve all 6 tiles in a single swap each, so we need at least 4 operations. One way to use exactly 4 operations is:

1. Swap the topmost two tiles in the middle column.
2. Flip the top right corner tile.
3. Swap the bottommost two tiles in the rightmost column.
4. Flip the middle tile of the leftmost column.

The picture below illustrates the states the floor goes through.



Additional Sample - Test Set 2

The following additional sample fits the limits of Test Set 2. It will not be run against your submitted solutions.

Sample Input

```
1
1 5 1000 1
MGGGG
GGGMM
```

Sample Output

```
Case #1: 1003
```

In the Sample Case for Test Set 2, flips are so expensive that we want to avoid them at all costs. We need at least one since our desired floor state has more magenta tiles than the current one, and swaps do not change that amount. We can do it optimally with just one flip like this:

1. Swap the leftmost two tiles.
2. Flip the rightmost tile.
3. Swap the second and third tiles from the left.
4. Swap the third and fourth tiles from the left.

The picture below illustrates all the states the floor goes through.

