2020 Team Formation Test (Round 1) Paper 1

Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
T2001	Alarm Clock	1.000 s	256 MB	23 + 24 + 23 + 30
T2002	Gradient Patterns	1.000 s	256 MB	13 + 87
T2003	Pac Man	1.000 s	256 MB	19 + 14 + 27 + 40

Notice:
Unless otherwise specified, inputs and outputs shall follow the format below:

- One space between a number and another number or character in the same line.
- No space between characters in the same line.
- Each string shall be placed in its own separate line.
- Outputs will be automatically fixed as follows: Trailing spaces in each line will be removed and an end-of-line character will be added to the end of the output if not present. All other format errors will not be fixed.

C++ programmers should be aware that using C++ streams (`cin` / `cout`) may lead to I/O bottlenecks and substantially lower performance.

For some problems 64-bit integers may be required. In Pascal it is `int64`. In C/C++ it is `long long` and its token for `scanf` /`printf` is `%lld`.

All tasks are divided into subtasks. You need to pass all test cases in a subtask to get points.

T2001 - ALARM CLOCK

Time Limit: 1.000 s / Memory Limit: 256 MB

Dr. Jones's company is developing a new smartphone called jPhone. In order to provide the best user experience and beat all the smartphones in the market, Dr. Jones has decided to design and implement their own operation system "jOS" and its pre-installed applications.

The software engineering team at Dr. Jones' company is current building the "Alarm Clock" application, for jOS. The application should allow users to create recurring alarms. For each of the recurring alarms being created, users have to configure the followings:

- The time to ring hh:mm (in 24-hour format). The alarm will ring at hh:mm:00 sharply.
- The day(s) of week the alarm should be activated. The users can set it as:
  - One of the seven days (Monday to Sunday); or
  - The special option "Weekdays", which indicates the alarm should be activated on every Monday to Friday.

You are hired as a software tester intern at Dr. Jones' company. Your mentor is developing a piece of software to simulate the clock and monitor the ringing activities. You are asked to implement a separated program to generate the expected ringing activities to facilitate the testing of the "Alarm Clock" application.

Formally, your program is given two integers N and Q, where N denotes the number of alarms being set in the application, and Q denotes the number of tests. Each of the tests is represented by a time in 24-hour format hh:mm and a day of week d (Monday to Sunday), and your program has to answer the question "When should the alarm first ring if the user sleeps at hh:mm:30 on the day d?". Note that the tests are independent from each other. In other words, the answers of the tests will not affect each other.

It is guaranteed that the N alarms are pairwise distinct. However, it is possible that two alarms are set to ring at the same moment (see Sample Test 4).

INPUT

The first line contains a single integer N, the number of alarms set.
In the next N lines, each line represents an alarm set. The line starts with the time in 24-hour format, represented by two 2-digit integers separated by a space: $hh\ mm$ ($00 \le hh \le 23$, $00 \le mm \le 59$), followed by the recurring day(s) of week option, which is one of the following 8 strings: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, `Sunday`, `Weekdays`. The time and the recurring day(s) of week option is separated by a space.

The next line contains a single integer Q, the number of tests.
In the next Q lines, each line represents a test. The line starts with the time in 24-hour format, represented by two 2-digit integers separated by a space: $hh\ mm$ ($00 \le hh \le 23$, $00 \le mm \le 59$), followed by a day of week d, which is one of the following 7 strings: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, `Sunday`. The time and string d is separated by a space.

OUTPUT

Output Q lines, the i^{th} line represents the answer for the i^{th} test, i.e. when will the alarm first ring.
Each line should start with the time in 24-hour format, represented by two 2-digit integers separated by a space: $hh\ mm$ ($00 \le hh \le 23$, $00 \le mm \le 59$), followed by a day of week, which should be one of the following 7 strings: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, `Sunday`. The time and the day of week should be separated by a space.

SAMPLE TESTS

	Input	Output
1	2 08 00 Saturday 08 00 Sunday 1 23 15 Saturday	08 00 Sunday

This sample is applicable to Subtask 1.

	Input	Output
2	2 08 00 Saturday 08 00 Sunday 1 08 00 Sunday	08 00 Saturday

This sample is applicable to Subtask 1.
Note that the test assumes the user sleeps at 08:00:30 on Sunday. Therefore the first alarm to ring after sleep is 08:00 on Saturday.

	Input	Output
3	3 09 30 Monday 06 00 Monday 23 15 Monday 5 00 00 Monday 05 59 Monday 06 00 Monday 09 45 Monday 23 30 Monday	06 00 Monday 06 00 Monday 09 30 Monday 23 15 Monday 06 00 Monday

This sample is applicable to Subtask 2.

	Input	Output
4	3 07 15 Monday 07 15 Weekdays 07 00 Friday 7 23 57 Sunday 23 59 Sunday 00 00 Monday 23 00 Wednesday 23 00 Thursday 07 07 Friday 21 00 Friday	07 15 Monday 07 15 Monday 07 15 Monday 07 15 Thursday 07 00 Friday 07 15 Friday 07 15 Monday

Note that although the alarms are guaranteed to be pairwise distinct, it is possible that two alarms ring at the same moment.

SUBTASKS

For all cases:
$1 \leq N \leq 11520$
$1 \leq Q \leq 300000$

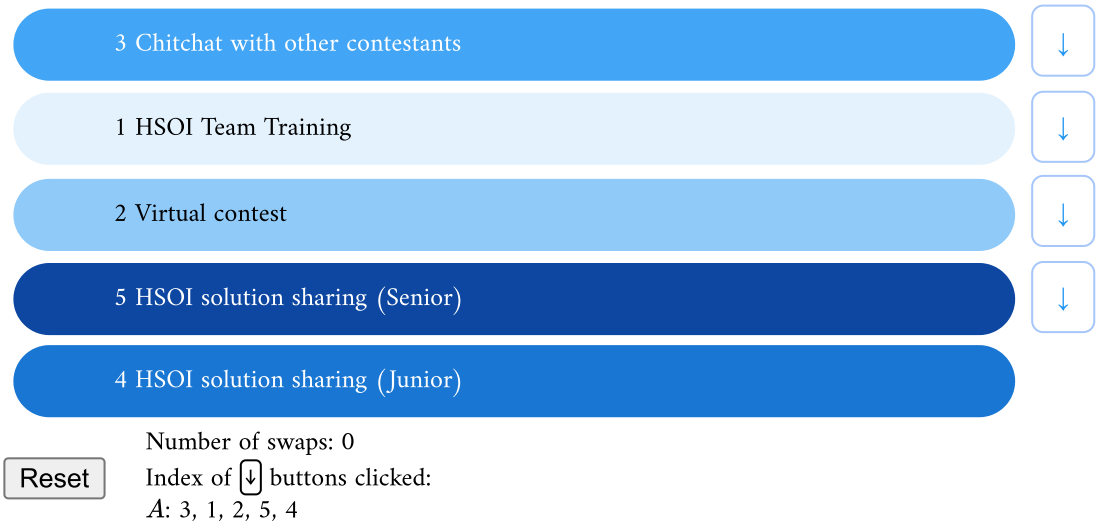
Points	Constraints

1	23	$N = 2$ $Q = 1$ The first alarm is `08 00 Saturday` The second alarm is `08 00 Sunday`
2	24	$1 \leq N, Q \leq 100$ All N alarms have their recurring option set as `Monday` All Q tests have $d =$ `Monday`
3	23	$1 \leq N, Q \leq 100$
4	30	No additional constraints

T2002 - GRADIENT PATTERNS

Time Limit: 1.000 s / Memory Limit: 256 MB

The Heung Shing Olympiad in Informatics offers N online activities during COVID-19 lockdown. Since the number of enrollment exceeds the capacity of a virtual meeting room, it decides to allocate one activity to each applicant based on their order of preferences. All applicants are required to submit an online form of this format to indicate their preferences:



In the form, the activities are listed one in a row, with different background colors. We number the activities based on the shade of the background colors — 1 is the activity with the lightest color, while N is the activity with the darkest color. The preference of an applicant forms a sequence A, listing the numbers of all activities from the top of the list, i.e. A is a permutation of $1...N$.

To rearrange the order of the activities, applicants can click on the ↓ button next to an activity. The form design is user-unfriendly that clicking on a ↓ button on a row only swaps activities on the same row and the next row, i.e. clicking on the i^{th} ↓ button only swaps values of A_i and A_{i+1}.

Alice realizes that some arrangements on the preference form beautiful gradient patterns on the list. She does not know how to create those patterns, so she asks you, her best friend, to help her out. You should tell her a sequence of ↓ clicks, so that she can the desired pattern from the given A.

In her opinion, she thinks that arrangements with exactly K 'peaks' are beautiful. There is a 'peak' on the i^{th} activity **if and only if** it has darker color than its adjacent activities, in other words **if and only if** the following conditions are all met:

- if $2 \leq i \leq N$, then A_i must be greater than A_{i-1}
- if $1 \leq i \leq N-1$, then A_i must be greater than A_{i+1}

Using the form above as an example, A is $3, 1, 2, 5, 4$ and it has 2 'peaks' on the 1^{st} and the 4^{th} element. To form a sequence with 1 'peak', for example, $1, 2, 3, 5, 4$, requires at least 2 swaps. For instance, Alice first clicks the 1^{st} ↓ button to swap the 1^{st} and the 2^{nd} element in A, then clicks 2^{nd} ↓ button to swap the 2^{nd} and the 3^{rd} element in A. You can try out this example, as well as other arrangements on the list above. The number of swaps, the indices of the ↓ button clicked, and the current A is summarized below the list as assistance. You can also restore the original A by clicking the `Reset` button.

When N is large, swapping activities can be tedious. To save Alice from clicking forever, she will rate your arrangements based on some criteria (See **Scoring** section below). Instruct her on how to form a beautiful pattern.

INPUT

The first line consists of two integers N and K.

The second line contains N integers A_1, A_2, ..., A_N, Alice's initial order of preference on the activities.

OUTPUT

Denote C be the number of swaps your solution requires to create K 'peaks' in A. Output 1 or 2 lines depending on the value of C.

If $C = 0$, output [0] on a single line.

If $C > 0$, output a single integer C on the first line. Then on the second line output C integers x_1, x_2, ..., x_C $(1 \leq x_i \leq N-1)$. In the i^{th} operation, Alice will click the x_i^{th} [↓] button, and the x_i^{th} order (A_{x_i}) would be swapped with the $(x_i+1)^{th}$ order (A_{x_i+1}).

If there are multiple solutions, output any of them.

SCORING

For each test case,

- You score 0% if the final sequence does not have K 'peaks' or $C > \frac{N(N-1)}{2} + \lfloor \frac{N-1}{2} \rfloor$; otherwise
- You score 100% if $C \leq \lfloor \frac{N(N-1)}{4} \rfloor$; otherwise
- You score 75% if $C \leq \lfloor \frac{N(N-1)}{4} \rfloor + \lfloor \frac{N-1}{2} \rfloor$; otherwise
- You score 60% if $C \leq \frac{N(N-1)}{2}$; otherwise
- You score 35% if $C \leq \frac{N(N-1)}{2} + \lfloor \frac{N-1}{2} \rfloor$.

You score for each subtask is the lowest score among all test cases within that subtask.

SAMPLE TESTS

	Input	Output
1	5 1 3 1 2 5 4	2 1 2

This corresponds to the example in the problem statement.
This output scores 100% as $C = 2 \leq 5$ $(5 = \lfloor \frac{5(5-1)}{4} \rfloor)$.

2	5 1 3 1 2 5 4	3 4 1 2

This is alternative solution to the example in the problem statement.
This output scores 100% as $C = 3 \leq 5$ $(5 = \lfloor \frac{5(5-1)}{4} \rfloor)$.

3	4 1 1 2 3 4	0

This is from subtask 1 as A is sorted in ascending order.
Print an empty line if no button clicks are required.
This output scores 100% as $C = 0 \leq 3$ $(3 = \lfloor \frac{4(4-1)}{4} \rfloor)$.

4

6 3 1 2 3 4 5 6	8 2 3 2 1 2 4 3 5

This is from subtask 1 as A is sorted in ascending order.
The final A is $4, 3, 5, 1, 6, 2$, and it has 3 'peaks' on the 1^{st}, the 3^{rd} and the 5^{th} activity.
This output scores 75% as $C = 8$, and $7 < 8 \leq 9$ ($7 = \lfloor \frac{6(6-1)}{4} \rfloor$, $9 = \lfloor \frac{6(6-1)}{4} \rfloor + \lfloor \frac{6-1}{2} \rfloor$).

SUBTASKS

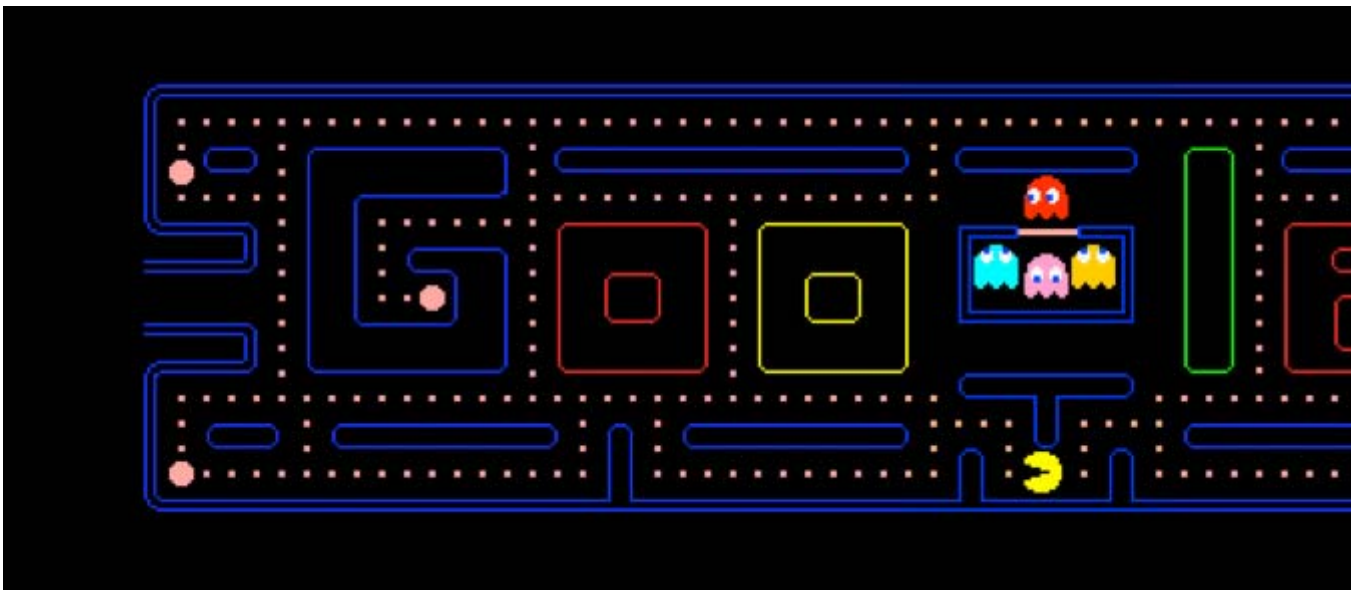
For all cases:
$1 \leq N \leq 300$
$1 \leq K \leq \lceil \frac{N}{2} \rceil$
$1 \leq A_i \leq N$, A is a permutation of $1, 2, \ldots, N$

	Points	Constraints
1	13	A is sorted in ascending order, in other words, $A_i = i$
2	87	No additional constraints

T2003 - PAC MAN

Time Limit: 1.000 s / Memory Limit: 256 MB

Do you know about the popular game “Pac Man”?



In the game, you act as the yellow Pac-man and your objective is to obtain as many Pac-dots (the small and big circles on the map) as possible while escaping from the ghosts (in the center) chasing around you.

You think that this game is too unfair – why is the Pac-man always chased by the ghosts? Therefore, as an experienced programmer, you are writing a new game that brings hope to the poor Pac-man. Originally, there are only 4 ghosts in the game, namely Blinky (red one), Pinky (pink one), Inky (blue one), and Clyde (orange one). However, this is certainly not flexible enough for Pac-man’s revenge! Therefore, you have decided to create N ghosts aligning in a straight line. The ghosts are indexed from left to right. The leftmost ghost is indexed 1 and the rightmost ghost is indexed N. The ghost with index i has a power level of A_i. Two ghosts might have the same power level.

The Pac-man also has a power level, which is C initially. During the game, it will walk through the 1st ghost, then the 2nd ghost, and eventually the N-th ghost in order to eat them all – just like Pac-man with power pellets in the original game! An eating action is considered successful if Pac-man's current power level is strictly greater than the ghost's power level. After a successful eating action, the Pac-man’s power level increases by the power level of the eaten ghost. For example, if Pac-man's current power level is 5 and the ghost's power level is 3, Pac-man's power level will become 8 after eating the ghost. And of course, for any unsuccessful eating action, it will be an immediate game-over and the Pac-man will no longer be allowed to continue with the game. Pac-man will win the game if he can successfully eat the N-th ghost.

After a few testing, you have realized that the game is probably too easy for Pac-man. Therefore, you have decided to modify the game a bit so that the Pac-man won’t be super overpowered. Now, for the FIRST eating action only, Pac-man's power level will decrease by the power level of the ghost, instead of increase by it.

Finally, for the sake of game balance, you will be satisfied if you can stop the Pac-man from winning by rearranging the order of the ghosts. Essentially, you want to find a proposal of rearrangement according to the following specification: The i-th operation moves the ghost with index X_i to the front (i.e. it becomes the new 1st ghost), where X_i is an index specified by you. (the indices here always refer to the initial indices before any rearrangements) You want to perform K such operations, such that Pac-man will lose the game at some point during the game. Note that for maximizing the program's efficiency, you would like to minimize the number of operations (K). If there are multiple rearrangement proposals with the same minimized K, then the lexicographically smallest operation sequence $X_1, X_2, \ldots, X_K$ is preferred. In case that there is no rearrangement that can make Pac-man loses, simply output `Pac-man is strong!`

INPUT

The first line contains 2 integers N and C, the total number of ghosts and the initial power level of Pac-man.

In the second line contains N integer. The i-th of which is A_i, the power level of the i-th ghost (index i).

OUTPUT

If there is no arrangement that can make Pac-man lose, output `Pac-man is strong!`.

Otherwise output an integer K in the first line, the minimum of operations to be performed such that Pac-man will lose the game.
Then output K more lines, where each line contains an integer that describes the operations in order. The integer in the i-th line should contain the index X_i chosen for the i-th operation. Notice that only the position but not the indices of the ghosts will be changed after any operation.

SCORING

For each test case:

- you will score 100% if your program
  - outputs the correct minimized K and the lexicographically smallest operation sequence of length K that makes Pac-man loses when it exists. **AND**
  - outputs `Pac-man is strong!` correctly.
- you will score 50% if your program
  - outputs the correct minimized K and any valid operation sequence of length K but not the lexicographically smallest one that makes Pac-man lose when it exists. **AND**
  - outputs `Pac-man is strong!` correctly.
- you will score 0 if your program
  - fails to output a valid operation sequence with K minimized that makes Pac-man lose when it exists. **OR**
  - fails to output `Pac-man is strong!` correctly.

You score for each subtask is the lowest score among all test cases within that subtask.

SAMPLE TESTS

	Input	Output
1	5 5 1 2 3 4 5	1 4

The optimal rearrangement will be moving the 4-th element to the front, which requires one operation only. The new array is 4 1 2 3 5. As Pac-man eats the first ghost, its power level becomes $5 - 4 = 1$. Since 1 (the new power level) is not greater than 1 (the power level of the 2nd ghost), Pac-man loses the game.

2

3 15 4 10 6	2 2 3

Please note that the final array is 6 10 4.
Although `2 3` and `3 2` are both possible solutions, `2 3` is the lexicographically smallest solution.

3

3 15 4 10 6	2 3 2

This output scores 50% as K is minimized and the sequence of operations is correct, but a lexicographically smaller solution exists.

4

5 10 1 2 1 2 3	Pac-man is strong!

It is impossible to make Pac-man lose using any kind of rearrangement.

SUBTASKS

For all cases:
$1 \leq N \leq 100000$
$1 \leq C, A_i \leq 10^7$

	Points	Constraints
1	19	$1 \leq N \leq 200$
2	14	$1 \leq N \leq 4000$
3	27	$A_1 \leq A_2 \leq \cdots \leq A_N$
4	40	No additional constraints.