
Scales

Amina has six coins, numbered from 1 to 6. She knows that the coins all have different weights. She would like to order them according to their weight. For this purpose she has developed a new kind of balance scale.

A traditional balance scale has two pans. To use such a scale, you place a coin into each pan and the scale will determine which coin is heavier.

Amina's new scale is more complex. It has four pans, labeled A, B, C, and D. The scale has four different settings, each of which answers a different question regarding the coins. To use the scale, Amina must place exactly one coin into each of the pans A, B, and C. Additionally, in the fourth setting she must also place exactly one coin into pan D.

The four settings will instruct the scale to answer the following four questions:

1. Which of the coins in pans A, B, and C is the heaviest?
2. Which of the coins in pans A, B, and C is the lightest?
3. Which of the coins in pans A, B, and C is the median? (This is the coin that is neither the heaviest nor the lightest of the three.)
4. Among the coins in pans A, B, and C, consider only the coins that are heavier than the coin on pan D. If there are any such coins, which of these coins is the lightest? Otherwise, if there are no such coins, which of the coins in pans A, B, and C is the lightest?

Task

Write a program that will order Amina's six coins according to their weight. The program can query Amina's scale to compare weights of coins. Your program will be given several test cases to solve, each corresponding to a new set of six coins.

Your program should implement the functions `init` and `orderCoins`. During each run of your program, the grader will first call `init` exactly once. This gives you the number of test cases and allows you to initialize any variables. The grader will then call `orderCoins()` once per test case.

- `init(T)`
  - `T`: The number of test cases your program will have to solve during this run. `T` is an integer from the range $1, \ldots, 18$.
  - This function has no return value.
- `orderCoins()`
  - This function is called exactly once per test case.
  - The function should determine the correct order of Amina's coins by calling the grader

functions `getHeaviest()`, `getLightest()`, `getMedian()`, and/or `getNextLightest()`.

  - Once the function knows the correct order, it should report it by calling the grader function `answer()`.
  - After calling `answer()`, the function `orderCoins()` should return. It has no return value.

You may use the following grader functions in your program:

- `answer(W)` — your program should use this function to report the answer that it has found.
  - `W`: An array of length 6 containing the correct order of coins. `W[0]` through `W[5]` should be the coin numbers (i.e., numbers from 1 to 6) in order from the lightest to the heaviest coin.
  - Your program should only call this function from `orderCoins()`, once per test case.
  - This function has no return value.
- `getHeaviest(A, B, C)`, `getLightest(A, B, C)`, `getMedian(A, B, C)` — these correspond to settings 1, 2 and 3 respectively for Amina's scale.
  - `A`, `B`, `C`: The coins that are put in pans A, B, and C, respectively. `A`, `B`, and `C` should be three distinct integers, each between 1 and 6 inclusive.
  - Each function returns one of the numbers `A`, `B`, and `C`: the number of the appropriate coin. For example, `getHeaviest(A, B, C)` returns the number of the heaviest of the three given coins.
- `getNextLightest(A, B, C, D)` — this corresponds to setting 4 for Amina's scale
  - `A`, `B`, `C`, `D`: The coins that are put in pans A, B, C, and D, respectively. `A`, `B`, `C`, and `D` should be four distinct integers, each between 1 and 6 inclusive.
  - The function returns one of the numbers `A`, `B`, and `C`: the number of the coin selected by the scale as described above for setting 4. That is, the returned coin is the lightest amongst those coins on pans A, B, and C that are heavier than the coin in pan D; or, if none of them is heavier than the coin on pan D, the returned coin is simply the lightest of all three coins on pans A, B, and C.

Scoring

There are no subtasks in this problem. Instead, your score will be based on how many weighings (total number of calls to grader functions `getLightest()`, `getHeaviest()`, `getMedian()` and/or `getNextLightest()`) your program makes.

Your program will be run multiple times with multiple test cases in each run. Let r be the number of runs of your program. This number is fixed by the test data. If your program does not order the coins correctly in any test case of any run, it will get 0 points. Otherwise, the runs are scored individually as follows.

Let Q be the smallest number such that it is possible to sort any sequence of six coins using Q weighings on Amina's scale. To make the task more challenging, we do not reveal the value of Q

here.

Suppose the largest number of weighings amongst all test cases of all runs is $Q + y$ for some integer y. Then, consider a single run of your program. Let the largest number of weighings amongst all T test cases in this run be $Q + x$ for some non-negative integer x. (If you use fewer than Q weighings for every test case, then $x = 0$.) Then, the score for this run will be $\frac{100}{r((x+y)/5+1)}$, rounded *down* to two digits after the decimal point.

In particular, if your program makes at most Q weighings in each test case of every run, you will get 100 points.

Example

Suppose the coins are ordered $3\ 4\ 6\ 2\ 1\ 5$ from the lightest to the heaviest.

Function call	Returns	Explanation
getMedian(4, 5, 6)	6	Coin 6 is the median among coins 4, 5, and 6.
getHeaviest(3, 1, 2)	1	Coin 1 is the heaviest among coins 1, 2, and 3.
getNextLightest(2, 3, 4, 5)	3	Coins 2, 3, and 4 are all lighter than coin 5, so the lightest among them (3) is returned.
getNextLightest(1, 6, 3, 4)	6	Coins 1 and 6 are both heavier than coin 4. Among coins 1 and 6, coin 6 is the lightest one.
getHeaviest(3, 5, 6)	5	Coin 5 is the heaviest among coins 3, 5 and 6.
getMedian(1, 5, 6)	1	Coin 1 is the median among coins 1, 5 and 6.
getMedian(2, 4, 6)	6	Coin 6 is the median among coins 2, 4 and 6.
answer([3, 4, 6, 2, 1, 5])		The program found the right answer for this test case.

Sample grader

The sample grader reads input in the following format:

- line 1: T —- the number of test cases
- each of the lines from 2 to $T + 1$: a sequence of 6 distinct numbers from 1 to 6: the order of the coins from the lightest to the heaviest.

For instance, an input that consists of two test cases where the coins are ordered $1\ 2\ 3\ 4\ 5\ 6$ and $3\ 4\ 6\ 2\ 1\ 5$ looks as follows:

```
2
1 2 3 4 5 6
3 4 6 2 1 5
```

The sample grader prints the array that was passed as a parameter to the `answer()` function.

Teams

There is a class of N students, numbered 0 through $N-1$. Every day the teacher of the class has some projects for the students. Each project has to be completed by a team of students within the same day. The projects may have various difficulty. For each project, the teacher knows the exact size of a team that should work on it.

Different students may prefer different team sizes. More precisely, student i can only be assigned to a team of size between $A[i]$ and $B[i]$ inclusive. On each day, a student may be assigned to at most one team. Some students might not be assigned to any teams. Each team will work on a single project.

The teacher has already chosen the projects for each of the next Q days. For each of these days, determine whether it is possible to assign students to teams so that there is one team working on each project.

Example

Suppose there are $N = 4$ students and $Q = 2$ days. The students' constraints on team sizes are given in the table below.

student	**0**	**1**	**2**	**3**
A	1	2	2	2
B	2	3	3	4

On the first day there are $M = 2$ projects. The required team sizes are $K[0] = 1$ and $K[1] = 3$. These two teams can be formed by assigning student 0 to a team of size 1 and the remaining three students to a team of size 3.

On the second day there are $M = 2$ projects again, but this time the required team sizes are $K[0] = 1$ and $K[1] = 1$. In this case it is not possible to form the teams, as there is only one student who can be in a team of size 1.

Task

You are given the description of all students: N, A, and B, as well as a sequence of Q questions — one about each day. Each question consists of the number M of projects on that day and a sequence K of length M containing the required team sizes. For each question, your program must return whether it is possible to form all the teams.

You need to implement the functions `init` and `can`:

- `init(N, A, B)` — The grader will call this function first and exactly once.
    - `N`: the number of students.

  - `A`: an array of length `N`: `A[i]` is the minimum team size for student i.
  - `B`: an array of length `N`: `B[i]` is the maximum team size for student i.
  - The function has no return value.
  - You may assume that $1 \le$`A[i]`$\le$`B[i]`$\le$`N` for each $i = 0, \ldots,$`N`-1.
- `can(M, K)` — After calling `init` once, the grader will call this function Q times in a row, once for each day.
  - `M`: the number of projects for this day.
  - `K`: an array of length `M` containing the required team size for each of these projects.
  - The function should return 1 if it is possible to form all the required teams and 0 otherwise.
  - You may assume that $1 \le$`M`$\le N$, and that for each $i = 0, \ldots,$`M`-1 we have $1 \le$`K[i]` $\le N$. Note that the sum of all `K[i]` may exceed N.

Subtasks

Let us denote by S the sum of values of `M` in all calls to `can(M, K)`.

subtask	**points**	N	Q	**Additional Constraints**
1	21	$1 \le N \le 100$	$1 \le Q \le 100$	none
2	13	$1 \le N \le 100,000$	$Q = 1$	none
3	43	$1 \le N \le 100,000$	$1 \le Q \le 100,000$	$S \le 100,000$
4	23	$1 \le N \le 500,000$	$1 \le Q \le 200,000$	$S \le 200,000$

Sample grader

The sample grader reads the input in the following format:

- line 1: `N`
- lines 2, …, `N` + 1: `A[i] B[i]`
- line `N` + 2: `Q`
- lines `N` + 3, …, `N` + `Q` + 2: `M` `K[0] K[1]` … `K[M - 1]`

For each question, the sample grader prints the return value of `can`.

Boxes with souvenirs

The last act of the IOI 2015 opening ceremony is in progress. During the opening ceremony, each team was supposed to receive a box with a souvenir from the host. However, all volunteers are so fascinated by the ceremony that they completely forgot about the souvenirs. The only person who remembers about the souvenirs is Aman. He is an enthusiastic volunteer and he wants the IOI to be perfect, so he wants to deliver all the souvenirs in the least amount of time.

The venue of the opening ceremony is a circle divided into L identical sections. The sections around the circle are numbered consecutively from 0 to $L-1$. That is, for $0 \le i \le L-2$, sections i and $i+1$ are adjacent, and also sections 0 and $L-1$ are adjacent. There are N teams at the venue. Each team is sitting in one of the sections. Each section may contain arbitrarily many teams. Some sections may even be empty.

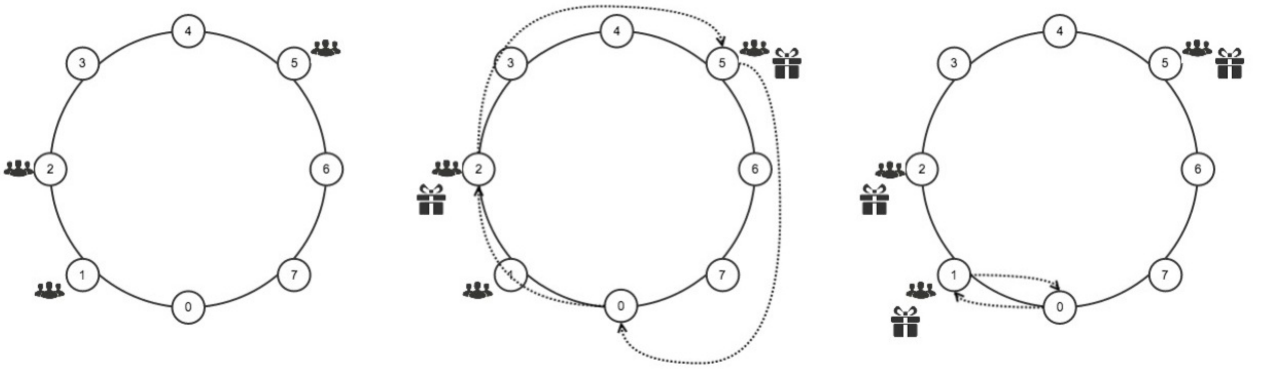
There are N identical souvenirs. Initially, both Aman and all of the souvenirs are in section 0. Aman should give one souvenir to each team, and after delivering the last souvenir he must return to section 0. Note that some teams may be sitting in section 0.

At any moment, Aman can only carry at most K souvenirs. Aman must pick up souvenirs in section 0, and this takes him no time. Each souvenir must be carried until it is delivered to one of the teams. Whenever Aman carries one or more souvenirs and reaches a section with a team that has not received a souvenir yet, he may give that team one of the souvenirs he carries. This also happens instantly. The only thing that takes time is movement. Aman can move around the circular venue in both directions. Moving to an adjacent section (either clockwise or counterclockwise) takes him exactly one second, regardless of how many souvenirs he carries.

Your task is to find the smallest number of seconds Aman needs to deliver all souvenirs and then return to his initial position.

Example

In this example we have $N = 3$ teams, Aman's carrying capacity is $K = 2$, and the number of sections is $L = 8$. The teams are located in sections 1, 2, and 5.



One of the optimal solutions is shown in the picture above. In his first trip Aman
takes two souvenirs, delivers one to the team in section 2, then the other to the team in section 5, and finally he returns to section 0. This trip takes 8 seconds. In his second trip Aman brings the remaining souvenir to the team in section 1 and then returns to section 0. He needs another 2 seconds to do this. Thus, the total time is 10 seconds.

Task

You are given N, K, L, and the positions of all teams. Compute the smallest number of seconds Aman needs to deliver all the souvenirs and to return to section 0. You need to implement the function `delivery`:

- `delivery(N, K, L, positions)` — This function will be called by the grader exactly once.
  - `N`: the number of teams.
  - `K`: the maximum number of souvenirs Aman can carry at the same time.
  - `L`: the number of sections in the venue of the opening ceremony.
  - `positions`: an array of length N. `positions[0], ..., positions[N-1]` give the section number of all teams. The elements of `positions` are in non-decreasing order.
  - The function should return the smallest number of seconds in which Aman can complete his task.

Subtasks

subtask	**points**	N	K	L
1	10	$1 \le N \le 1{,}000$	$K = 1$	$1 \le L \le 10^9$
2	10	$1 \le N \le 1{,}000$	$K = N$	$1 \le L \le 10^9$
3	15	$1 \le N \le 10$	$1 \le K \le N$	$1 \le L \le 10^9$
4	15	$1 \le N \le 1{,}000$	$1 \le K \le N$	$1 \le L \le 10^9$
5	20	$1 \le N \le 10^6$	$1 \le K \le 3{,}000$	$1 \le L \le 10^9$

subtask	**points**	N	K	L
6	30	$1 \leq N \leq 10^7$	$1 \leq K \leq N$	$1 \leq L \leq 10^9$

Sample grader

The sample grader reads the input in the following format:

- line 1: `N K L`
- line 2: `positions[0]` … `positions[N-1]`

The sample grader prints the return value of `delivery`.