Bases Conversion (23)

Your professor has just introduced you to the different bases in which a number can be written and wants to test your understanding. His favourite number is 23 (this should not surprise you too much... it is a prime number!) and for this reason his favourite bases are 2 and 3.

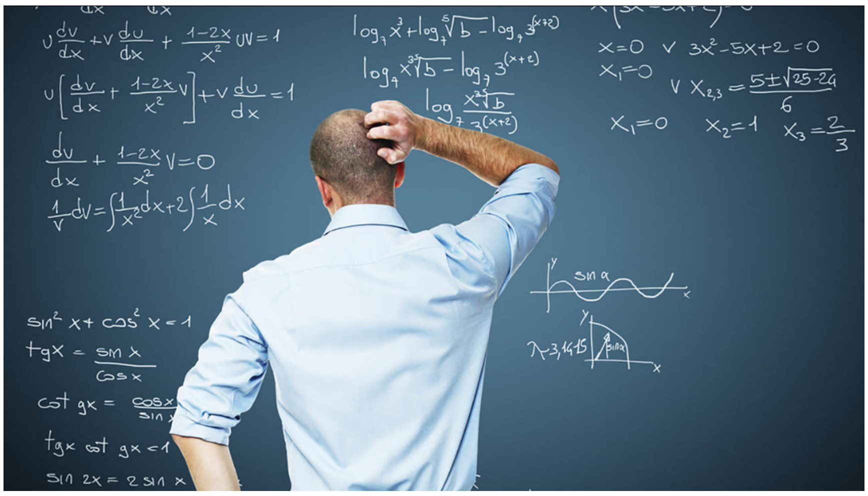


Figure 1: A professor figuring out how to keep students busy.

A number is called **special** if and only if the sum of the digits of its base *two* representation is the same as the sum of the digits of its base *three* representation.

Consider the number $6_{(10)}$ (the subscript denotes the basis) $= 110_{(2)} = 20_{(3)}$. The sum of the digits in base two is $1 + 1 + 0 = 2$, which is exactly the sum of the digits in base three: $2 + 0 = 2$. Thus, number $6_{(10)}$ is special. On the contrary, the number $9_{(10)} = 1001_{(2)} = 100_{(3)}$ is not special (the sums of digits are 2 and 1, respectively).

In order to keep the class busy without much work on his side, your professor has come up with a boring homework assignment. He has given you a list of T numbers $N_1, \ldots, N_T$ and for each of those he wants you to compute the number of special numbers from 1 to N_i (extremes included). Automate this tedious task by writing a program!

☞ Among the attachments of this task you may find a template file `23.*` with a sample incomplete implementation.

Input

The first line contains the integer T. The second line contains T integers N_i.

Output

You need to write a single line with T integers, where the i-th indicates the amount of special numbers in the range between 1 and N_i.

Constraints

- $1 \leq T \leq 10\,000$.
- $1 \leq N_i \leq 100\,000\,000$ for each $i = 0 \ldots T-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (30 points)	The sum of all N_i does not exceed $1\,000\,000$.
– **Subtask 3** (35 points)	$N_i \leq 1\,000\,000$ for each $i = 0 \ldots T-1$.
– **Subtask 4** (35 points)	No additional limitations.

Examples

input	output

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

`1`
`10`
	`4`

Explanation

The representations in base two and three of the numbers between 1 and 10 are contained in the following table. Special numbers have been highlighted in yellow.

Base 10	Base 2	Base 3	Sum in base 2	Sum in base 3
1	1	1	1	1
2	10	2	1	2
3	11	10	2	1
4	100	11	1	2
5	101	12	2	3
6	110	20	2	2
7	111	21	3	3
8	1000	22	1	4
9	1001	100	2	1
10	1010	101	2	2

In the **first sample case** for the ranges up to 1, 2 and 5 there is only a single special number ($1_{(10)}$). If we consider numbers up to 6, there are two special numbers ($1_{(10)}$ and $6_{(10)}$).

In the **second sample case**, in the range up to $10_{(10)}$ there are four special numbers (the ones highlighted in the table above).

Kill Those Bugs! (`blindpunch`)

In the middle of the night Luca has been attacked by bugs, which are very well organised and form a line in front of his bed. In order to protect himself, Luca has K slippers to be thrown at the bugs one at a time (but optionally he can throw multiple slippers at the same bug). The bugs, being very creative, are playing dead to fool him: the room is dark and he can only barely see the bugs, without knowing which ones are squashed and which are alive.

Figure 1: Kill those damn bugs!

Fortunately, Luca knows that the i-th bug in the line of N bugs has a probability P_i to be squashed when he throws a slipper at it. That probability does not change, meaning that subsequent slippers also have a probability P_i to kill it, if it is not already killed.

☞ A quick refresh on how probability works: if a bug has a probability P_i to be killed, obviously the first time you throw a slipper at it it has P_i probability to die. If you throw a second slipper at it, it can die only if it has not died during the first attempt, which happens with probability $1 - P_i$. Thus, it dies at the second attempt with probability $P_i \cdot (1 - P_i)$. If you still want to throw a third slipper at it, it has a probability of only $(1 - P_i)^2$ of being still alive, thus the third slipper can kill it with probability $P_i \cdot (1 - P_i)^2$. In general, the probability to die at the n-th slipper is $P_i \cdot (1 - P_i)^n$. Check out the examples for additional clarifications.

Luca is very tired and hates bugs crawling around in his room in the middle of the night. Help him use the slippers in the wisest way possible, in order to maximize the expected number of killed bugs.

☞ Among the attachments of this task you may find a template file `blindpunch.*` with a sample incomplete implementation.

✍ Warning: as this task deals with floating point numbers, it is highly recommended to use the attached templates for a correct I/O.

Input

The first line contains an integer T, the number of scenarios that you have to solve. The next $2T$ lines describe the scenarios. Each scenario is described with a first line containing two integers, N and K, respectively the number of bugs and the number of slippers; and a second line which contains N floating point numbers P_i, the probability of dying for the i-th bug.

Output

You need to write a line with a floating point number for each different scenario: the expected value of killed bugs.

Constraints

- $1 \leq N, T \leq 200\,000$.
- $1 \leq K \leq 1\,000\,000$.
- $0 \leq P_i \leq 1$ for each $i = 0 \ldots N-1$.
- The sum of the bugs across all scenarios does not exceed $200\,000$.
- The sum of the slippers across all scenarios does not exceed $1\,000\,000$.
- Floating points probabilities in input have at most 9 decimals and you must print the result truncating it down at 6 decimal places (the attached templates already implement these steps).

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (40 points)	T, the sum of the bugs and the sum of the slippers across all scenarios does not exceed 500.
– **Subtask 3** (30 points)	T, the sum of the bugs and the sum of the slippers across all scenarios does not exceed 5 000.
– **Subtask 4** (30 points)	No additional limitations.

Examples

input	output
`1`	`0.900000`
`4 2`	
`0.1 0.1 0.5 0.4`	
`1`	`2.150000`
`5 4`	
`0.8 0.6 0.5 0.2 0.1`	

Explanation

In the **first sample case**, the best strategy is to throw the first slipper at the third bug (0.5 of probability of killing it). Then, you should throw the second slipper at the fourth bug (0.4 of probability of killing it). Note that you would still have 50% of chance of killing the third bug, but in half cases you would have already killed it with the first slipper (the "cumulated" chance for that bug at the second slipper would be just 0.25). Overall, the expected value is $0.5 + 0.4 = 0.9$.

In the **second sample case**, a possible strategy to maximize the number of killed bugs is to throw the first three slippers at the first three bugs, yielding an expected value of $0.8 + 0.6 + 0.5 = 1.9$. The fourth slipper can then be thrown again at the third bug, which still has a 50% chance of being alive and thus contributes 0.25 to the expected value.

Save The Barrels (butoaie)

A swarm of bugs from Bugland invaded William's wine cellar trough a magic portal. The wine cellar consists of N rooms, each full of barrels of wine, with the i-th room being infested by V_i bugs. William has at his disposal N insecticide diffusers of two types: K *AntiBug* which remove P bugs a day, and $N - K$ *ZeroBugs*, which remove Q bugs a day.

Figure 1: Different brands of insecticide diffusers.

Every day, at most one diffuser can be used in every room without compromising the quality of the wine. What is the minimum number of days needed to save the barrels of wine by eliminating all the bugs?

☞ Among the attachments of this task you may find a template file `butoaie.*` with a sample incomplete implementation.

Input

The first line contains integers N and K. The second line contains integers P and Q. The third line contains N integers, the i-th of those represents the number of bugs V_i present in the i-th room.

Output

You need to write a single line with an integer: the minimum number of days necessary to eliminate all the bugs.

Constraints

- $K \leq N \leq 200\,000$.
- $P, Q \leq 10^9$.
- $V_i \leq 10^9$ for each $1 \leq i \leq N$.
- At most one diffuser can be used in a single room each day.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (20 points)	$K = N$.
– **Subtask 3** (30 points)	$K \le N \le 10\,000$, $P, Q \le 100$ and $V_i \le 10\,000$.
– **Subtask 4** (50 points)	No additional limitations.

Examples

input	output

5 2
3 1
3 4 5 7 8
	4

Explanation

In the **first sample case** this is what happens:

- After one day: 3 4 5 7 8 $\to$ 2 3 4 4 5
  - 3 bugs are eliminated from rooms 4 and 5;
  - 1 bug is eliminated from rooms $1, 2$ and 3.
- After two days: 2 3 4 4 5 $\to$ 1 2 3 1 2
  - 3 bugs are eliminated from rooms 4 and 5;
  - 1 bug is eliminated from rooms $1, 2$ and 3.
- After three days: 1 2 3 1 2 $\to$ 0 1 0 0 0
  - 3 bugs are eliminated from room 3;
  - 2 bugs are eliminated from room 5;
  - 1 bug is eliminated from rooms $1, 2$ and 4.
- After four days: 0 1 0 0 0 $\to$ 0 0 0 0 0
  - 1 bug is eliminated from rooms 2.

Fibonacci Colonies (`fibonaccibug`)

Bug colonies have been the center of attention of scientists for a long time. Through some technological advancements, we are now able to describe a bug colony using a number known as the *degree* of the colony. A colony of degree 0 or 1 represents a colony with one bug. A colony of degree $i > 1$ is obtained by merging a colony of degree $i - 1$ together with a colony of degree $i - 2$. As such, a colony of degree 2 has two bugs, a colony of degree 3 has three bugs, a colony of degree 4 has five bugs and so on.

Marco owns the biggest bug farm in the world, having at his disposal a virtually infinite amount of colonies of any degree. Every day he receives N offers, each described by two numbers A_i and B_i, meaning that he can sell as many colonies of degree A_i as he wants and get B_i money for each colony of that degree. Unfortunately, the antitrust laws on the bug trading market forbid him to sell more than K bugs in a single day overall (selling a colony is equivalent to selling all the bugs in that colony). Given the description of T days, if he optimally chooses which offers to accept, what is the maximum amount of money Marco can obtain in each day?

☞ Among the attachments of this task you may find a template file `fibonaccibug.*` with a sample incomplete implementation.

Input

The first line contains one integer T, the number of days. The following lines contain the description of each day. For each day, the first line contains two integers N and K, the number of offers and the maximum number of bugs you can sell that day. The following N lines contain contain two integers A_i and B_i, the colony of the offer and the price per colony.

Output

You need to write T lines, each with an integer: the maximum profit you can make for each day.

Constraints

- $1 \le T,\ N,\ K \le 100\,000$.
- $0 \le A_i \le 100\,000$.
- $1 \le B_i \le 10^9$.
- The sum of all N and all K across the days of a single input does not exceed $201\,000$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (10 points)	$T = 1,\ N \le 6,\ K \le 6,\ A_i \le 10$.
– **Subtask 3** (10 points)	$K = 1$.
– **Subtask 4** (35 points)	$N, K \le 5500$ and $A_i \le 11\,000$.
– **Subtask 5** (45 points)	No additional limitations.

Examples

input	output

1
5 11
1 2
2 2
3 5
4 9
5 50
	56

2
3 10
1 10
4 60
3 40
2 10
1 30
2 40
	130
	300

Explanation

In the **first sample case** it is optimal to choose the fifth offer once and the first one three times.

In the **second sample case**, for the first day it is optimal to choose the first offer once and the third offer three times; for the second day it is optimal to choose ten times the first offer.

Funny Circuits (`funnygraph`)

Marco is studying electronics at *Politecnico di Milano* and his teacher assigned him an experimental project to do at the lab. The project is pretty simple: he has a circuit board with N connection points, and he should connect M components to those points, in the given order from component 0 to $M-1$. Each component i has a characteristic integral *potential difference* z_i and two *terminals*, which have to be connected to two given connection points a_i and b_i.

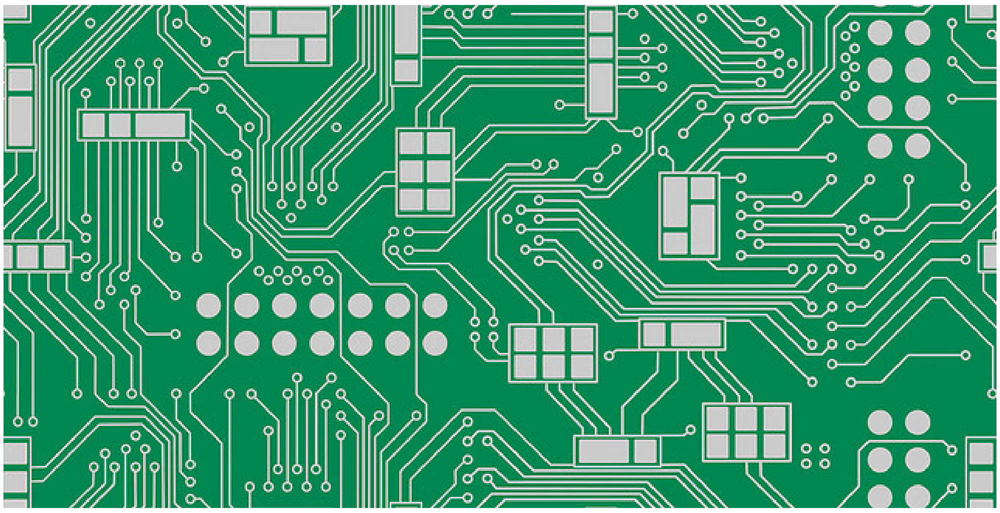


Figure 1: An example of an empty circuit board.

Once a circuit is built, each of the connection points $j = 0 \dots N-1$ naturally acquires a *potential* P_j. Furthermore, the i-th component can work correctly only provided that the difference of the potentials of its terminals is exactly z_i: in formulas, if $P_{a_i} - P_{b_i} = z_i$ (or equivalently $P_{b_i} - P_{a_i} = -z_i$: the orientation of components matters). The laws of electromagnetism ensure that whenever an assignment of potentials satisfying the potential differences of components exists, such an assignment will naturally arise. However, if the plan of the circuit is wrong, such an assignment may not exist!

Marco has carefully followed the instructions from his teacher, but sadly, the circuit is not behaving as expected. In fact, he suspects that some of the last components broke the circuit, and should actually be removed in order for the project to succeed. Help Marco by computing the largest number K so that a compatible assignment of potentials exist for components $i = 0 \dots K-1$.

☞ Among the attachments of this task you may find a template file `funnygraph.*` with a sample incomplete implementation.

Input

The first line contains two integers N and M. The next M lines contain integers a_i, b_i and z_i describing the components in the order they should be added to the circuit.

Output

Your program has to write a single integer: the maximum number K of components Marco can add.

Constraints

- $1 \le N \le 100\,000$.
- $1 \le M \le 200\,000$.
- $-10^9 \le z_i \le 10^9$ for each $i = 0 \ldots M-1$.
- $0 \le a_i, b_i \le N-1$ for each $i = 0 \ldots M-1$.
- There could be components with both terminals connected to the same node ($a_i = b_i$).
- There could be more than one component between the same pair of connection points.
- A compatible assignment for all $K = M$ components may exist: maybe the plan is correct and Marco just failed to implement it!

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (10 points) $a_i = b_i$ for all $i = 0 \ldots N-1$.

– **Subtask 3** (15 points) $N, M \le 5$, $z_i = 1$ for all $i = 0 \ldots N-1$.

– **Subtask 4** (10 points) $N, M \le 50$.

– **Subtask 5** (30 points) $N, M \le 2000$.

– **Subtask 6** (25 points) $M \le 100\,000$.

– **Subtask 7** (10 points) No additional limitations.

Examples

input	output

4 5
0 1 1
1 3 -1
3 0 0
3 2 4
2 0 3
	4

3 3
0 1 1
2 2 0
1 0 3
	2

Explanation

In the **first sample case**, ignoring the last component, the *potentials* can be 10, 9, 6 and 10. Note that this assignment is not unique, in fact adding or subtracting any value from all the points doesn't change the solution. It's not possible to add the last component, so the answer should be 4 components.

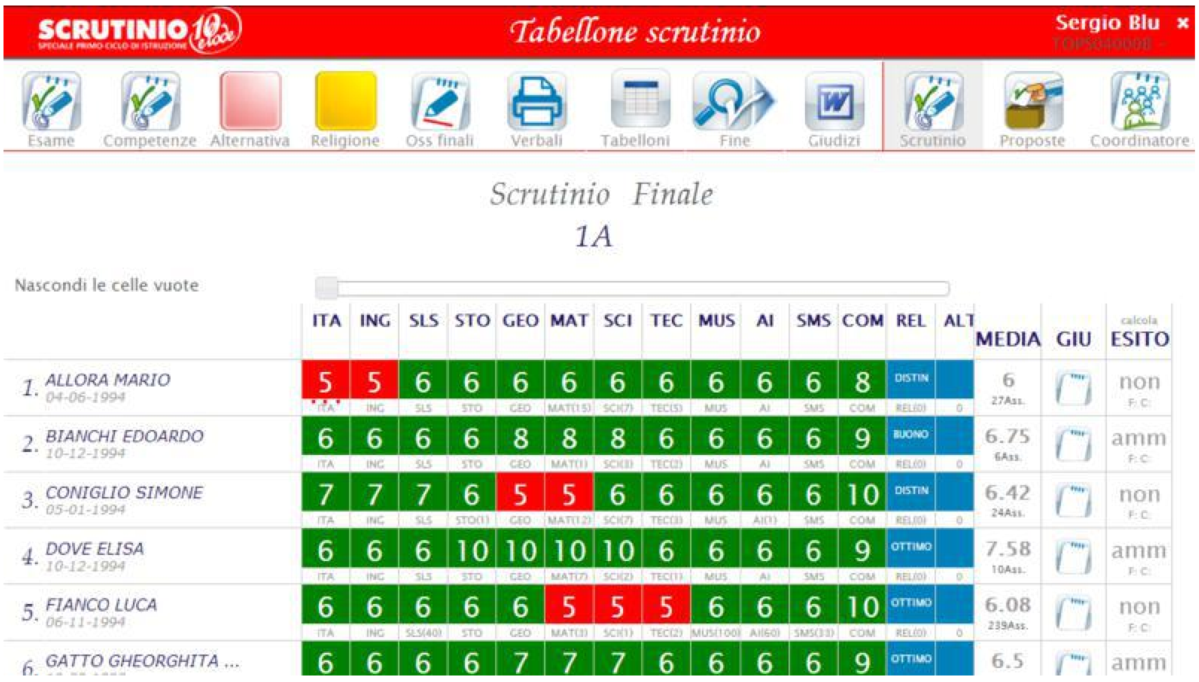
In the **second sample case** a possible assignment, ignoring the last component, is 5, 4 and any value for the last point. The first component requires that P_1 is one less of P_0, the second component just requires that $P_2 = P_2$. The last component is incompatible with the first one, so it cannot be inserted.

Improving Grades (grades)

In Italy, every parent of a high-school student can check how their child is doing in terms of grades. In fact, at any given moment the parent can access the "electronic register" where the professors regularly upload the grades obtained by their students in the various tests throughout the year.

Edoardo is in high-school, and he's definitely not the best of his class: he regularly gets lots of insufficient grades! Grades in Italian high-schools go from 2 to 10, and a grade is said to be "sufficient" when it is **higher than or equal to 6**.

Luckily, Edoardo's parents are not very tech-savvy: they know how to use a tablet, but they can't access the "electronic register" website on their own. Edoardo thinks this is a good thing, since all of his friends are constantly kept in check by their parents.



Scrutinio Finale
1A

	ITA	ING	SLS	STO	GEO	MAT	SCI	TEC	MUS	AI	SMS	COM	REL	ALT	MEDIA	GIU	calcola ESITO
1. ALLORA MARIO 04-06-1994	5	5	6	6	6	6	6	6	6	6	6	8	DISTIN	0	6 27Ass.		non F: C:
2. BIANCHI EDOARDO 10-12-1994	6	6	6	6	8	8	8	6	6	6	6	9	BUONO	0	6.75 6Ass.		amm F: C:
3. CONIGLIO SIMONE 05-01-1994	7	7	7	6	5	5	6	6	6	6	6	10	DISTIN	0	6.42 24Ass.		non F: C:
4. DOVE ELISA 10-12-1994	6	6	6	10	10	10	10	6	6	6	6	9	OTTIMO	0	7.58 10Ass.		amm F: C:
5. FIANCO LUCA 06-11-1994	6	6	6	6	6	5	5	5	6	6	6	10	OTTIMO	0	6.08 239Ass.		non F: C:
6. GATTO GHEORGHITA ...	6	6	6	6	7	7	7	6	6	6	6	9	OTTIMO		6.5		amm

Figure 1: The "electronic register".

The end of the year is getting close so, today, Edoardo's parents asked him to show them his grades. He thought about cheating (faking the website content) but he would feel bad about it. Instead, he wants to impress his parents by showing them that he *improved* during the year.

For this reason Edoardo decided to *zoom in* in the long list of grades, so that only part of the list is visible. More specifically: Edoardo can hide a prefix and a suffix of the list by zooming in.

To impress his parents as much as possible, Edoardo wants this "new list" to show a **clear improvement**: the first grade must be *insufficient* and the last one should be *sufficient*. In order to make the "new list" look realistic, he should also try to hide as few grades as possible.

Help Edoardo measure the maximum number of grades he can include in a zoomed-in list so that the first grade is insufficient and the last one is sufficient. If this is impossible, return `-1`.

☞ Among the attachments of this task you may find a template file `grades.*` with a sample incomplete implementation.

Input

The first line contains the only integer N, the total number of grades uploaded by Edoardo's teacher. The second line contains N integers G_i, the grades.

Output

You need to write a single line with an integer: the maximum number of grades that can be included in the zoomed-in list, or `-1` if it's not possible to find a suitable list.

Constraints

- $1 \le N \le 100\,000$.
- $2 \le G_i \le 10$ for each $i = 0 \ldots N-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (30 points) $N \le 10$.

– **Subtask 3** (50 points) $N \le 1000$.

– **Subtask 4** (20 points) No additional limitations.

Examples

input	output

`5`
`7 5 7 8 4`
	`3`

`6`
`7 5 7 8 4 6`
	`5`

`3`
`6 7 4`
	`-1`

Explanation

In the **first sample case**, it is possible to obtain the zoomed-in list `5 7 8` of length 3, which starts with an insufficient grade and ends with a sufficient grade. It's impossible to find a longer list.

In the **second sample case**, at the cost of "ending with a 6" instead of with an 8, Edoardo will prefer zooming in the sublist `5 7 8 4 6` because having 5 grades in the electronic register is more realistic than having just 3.

In the **third sample case** Edoardo cannot show that he improved, since the only sublist starting with an insufficient grade is `4`, which does not end with a sufficient grade.

Compulsive Smartphone Shopping (smartphone)

Edoardo is very passionate about technology, especially about smartphones. Unfortunately, he just dropped his phone and the screen shattered, so he needs to buy another one as soon as possible!

Edoardo is going to visit N shops in order. Every time he enters a new shop, he looks for the phone with the highest price and, if it is strictly more valuable than all of the smartphones he currently has, he buys it. Edoardo doesn't skip any shop, even if he has already purchased a smartphone elsewhere (he wants only the very best!).

You know that the most valuable smartphone in the i-th shop costs P_i. You also know that the initial smartphone has no value, since it is broken. How much will Edoardo spend in total?

☞ Among the attachments of this task you may find a template file `smartphone.*` with a sample incomplete implementation.

Input

The first line contains the only integer N. The second line contains N integers P_i.

Output

You need to write a single line with an integer: the total cost paid by Edoardo.

Constraints

- $1 \leq N \leq 100\,000$.
- $1 \leq P_i \leq 10^9$ for each $i = 0 \ldots N-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.
- **Subtask 2** (10 points) $N \leq 10$, $P_i \leq 1000$.
- **Subtask 3** (20 points) $N \leq 1000$, $P_i \leq 1000$.
- **Subtask 4** (30 points) $P_i < P_{i+1}$ for every $i = 0 \ldots N-2$.
- **Subtask 5** (40 points) No additional limitations.

Examples

input	output
5 1 3 2 2 5	9
4 1 3 5 7	16

Explanation

In the **first sample case** Edoardo buys a smartphone in the first, second and fifth shop, paying 9 in total.

In the **second sample case** Edoardo buys a smartphone in every shop, paying a total of 16.

Ant Supercolonies (sprei)

Giorgio's house has a huge bugs problem: N colonies of ants are infesting the place, and he must kill them once and for all! Since he is a very abstract person, Giorgio lives in a M-dimensional cube of side B, so that the position of every ant colony can be represented as an array of coordinates $(A_1, A_2, \ldots, A_M)$ with integer values ranging from 0 to $B - 1$.

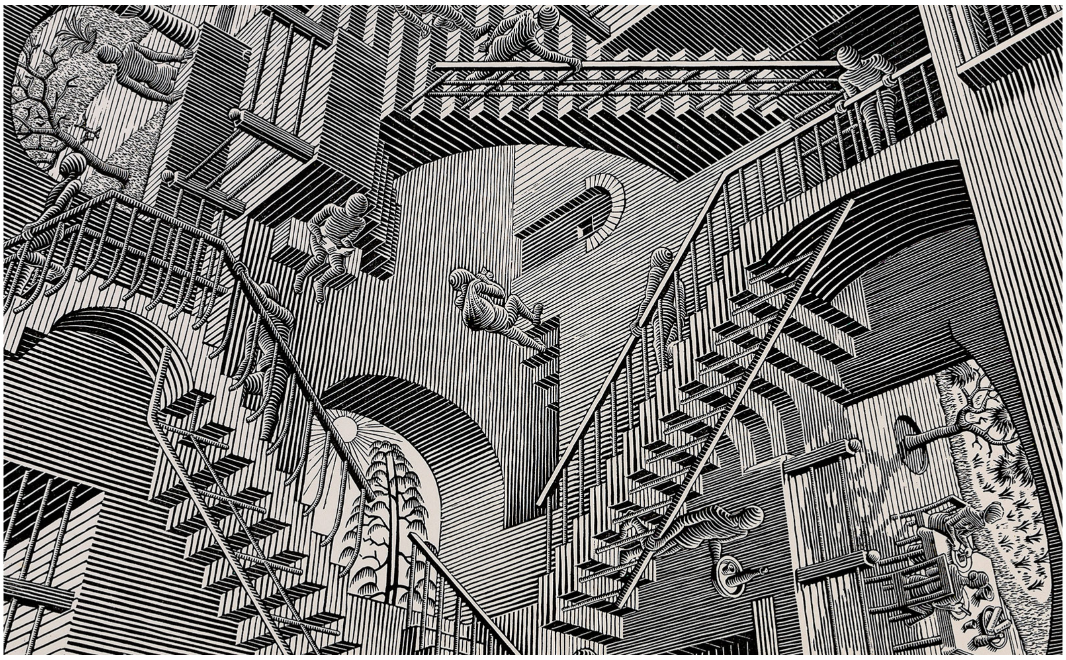


Figure 1: Panoramic painting of Giorgio's house: such corners, much bugs!

Since ants are social entities, they always try to unify the colonies. Thereby, if two colonies are adjacent in the house (the position of one is obtained by adding or subtracting 1 from a single coordinate of the position of the other), they build a tunnel among them, creating a supercolony unifying the two smaller colonies and exacerbating the infestation power by orders of magnitude.

Giorgio knows the positions of the N colonies, and can use pest control for destroying any one of them. However, the pest control process is tedious and expensive, so Giorgio would like to destroy the *smallest* number of colonies that guarantees no super-colony can be created among the remaining ones. How many does he need to destroy at least?

☞ Among the attachments of this task you may find a template file `sprei.*` with a sample incomplete implementation.

Input

The first line contains integers N, M and B. The following N lines contain M integers A_i, representing the position of a colony.

Output

You need to write a single line with an integer: the minimum number of colonies to be destroyed in order to prevent supercolonies.

Constraints

- $1 \leq N \leq 10\,000$.
- $1 \leq M \leq 100$.
- $1 \leq B \leq 10^9$.
- It is guaranteed that any two colonies have distinct positions.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (10 points)	$N = B^M$ (the house is full of bugs).
– **Subtask 3** (20 points)	$M = 1$ (the house is a line).
– **Subtask 4** (30 points)	$N \leq 10$.
– **Subtask 5** (40 points)	No additional limitations.

Examples

input	output
`5 3 4`	`2`
`1 1 1`	
`1 1 2`	
`1 1 3`	
`1 2 1`	
`1 2 3`	
`6 2 4`	`2`
`1 0`	
`0 1`	
`1 1`	
`2 1`	
`3 1`	
`2 2`	

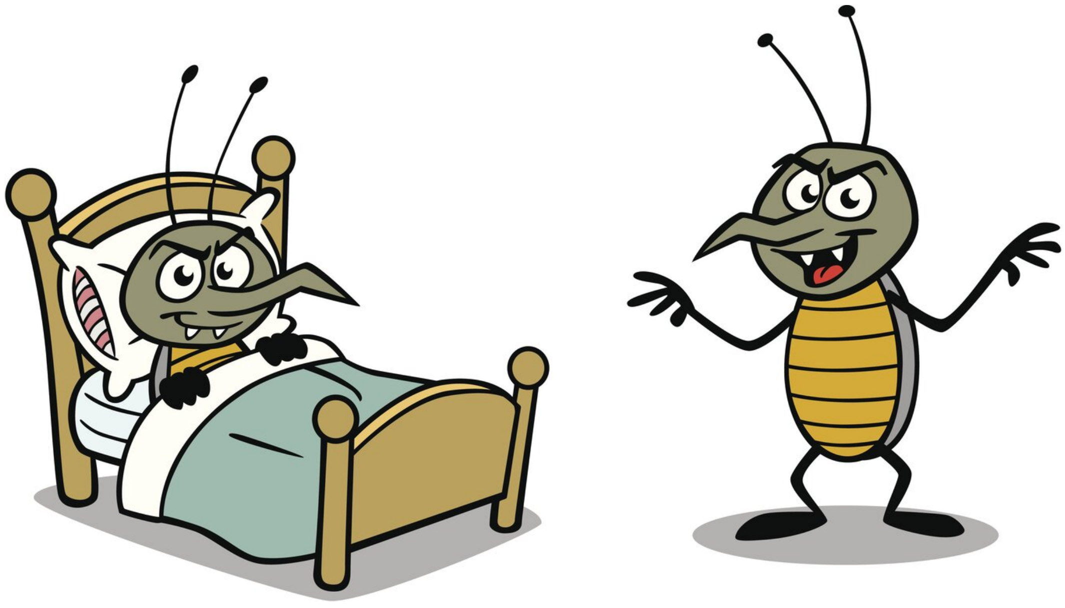
Explanation

In the **first sample case**, it suffices to destroy two colonies: the first and the third ones. Destroying a single colony is not enough: in order to prevent the last colony to form a supercolony, we need to destroy the third one, and that is not preventing the first two colony to gather into a supercolony.

In the **second sample case**, the optimal strategy is to destroy colonies $(1, 1)$ and $(2, 1)$.

Secret Meeting (teleport2)

Everyone knows that in Bugland rooms are an infinite plane surface in which each point is described by two coordinates (x, y): in particular, two bugs A and B are now waiting at coordinates (X_a, Y_a) and (X_b, Y_b). It is well known that in Bugland beds are perfect circles, described by the coordinates of their center and their radius: in particular, in the mentioned room there is a single bed with center $C = (X_c, Y_c)$ and radius R.



A and B want to have a secret meeting to plan the invasion of the room: thus, they need to move as silent as possible, according to any trajectory, and gather this way to a common meeting point (X_m, Y_m) (not necessarily of integer coordinates). Walking on the floor one unit makes 1 NU (Noise Unit): for example, walking from $(1, 1)$ to $(3, 1)$ makes 2 NU, and walking from $(1, 1)$ to $(3, 2)$ makes $\sqrt{5}$ NU. On the other hand, walking on the bed is completely silent. Help the two bugs A and B meet, using a path allowing the sum of their NUs to be minimal.

☞ Among the attachments of this task you may find a template file `teleport2.*` with a sample incomplete implementation.

✍ Warning: as this task deals with floating point numbers, it is highly recommended to use the attached templates for a correct I/O.

Input

The first line contains one integer T, the number of scenarios that you have to solve. Each of the following T lines describe a scenario and contain seven integers: X_a, Y_a, X_b, Y_b, X_c, Y_c and R.

Output

You need to write T lines, one for each scenario, each with a single floating point: the minimal NU made by the two bugs. The printed numbers must have **exactly** 6 decimals, rounded down.

Constraints

- $1 \le T \le 10\,000$.
- $-100\,000 \le X_a, Y_a, X_b, Y_b, X_c, Y_c \le 100\,000$.
- $0 \le R \le 100\,000$.
- There are no constraints on the meeting point and trajectories of the bugs!

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (10 points)	$R = 0$.
– **Subtask 3** (10 points)	$X_a = X_b = X_c$.
– **Subtask 4** (20 points)	A, B and C are collinear.
– **Subtask 5** (25 points)	The answer for every scenario is guaranteed to be an integer.
– **Subtask 6** (35 points)	No additional limitations.

Examples

input	output
`1`	`2.000000`
`-1 0 1 0 0 3 1`	

Explanation

In the **first sample case**, the first bug is in $(-1, 0)$, the second bug in $(1, 0)$ and the bed has radius 1 and center in $(0, 3)$. One of the possible solution is that the two bugs meet at $(0, 0)$.