

Problem A. All the Trades Are the Best

Input file: `standard input`
Output file: `standard output`
Time limit: 4 seconds
Memory limit: 512 megabytes

You are a trader at an investment company. One day, you discover that the stock you're responsible for trading follows an exactly repeating daily pattern: the day is divided into n equal time intervals, and the stock's price during the i -th interval is always a_i . As the n -th interval of one day concludes, the first interval of the next day immediately follows, with the price shifting from a_n to a_1 . This is a once-in-a-lifetime opportunity — if you can just buy and sell according to this pattern you've already figured out, during your working hours, there's a guaranteed way to profit!

Your working hours vary, and a working period may extend past midnight. You are given q working periods. The i -th period starts at slot l_i and ends at slot r_i , including both endpoints. The slots in chronological order are:

- $l_i, l_i + 1, \dots, r_i$ if $l_i \leq r_i$;
- $l_i, l_i + 1, \dots, n, 1, 2, \dots, r_i$ if $l_i > r_i$.

In each working period, you may complete at most k transactions. Each transaction consists of buying one share and selling it in a later time slot, with profit equal to the sale price minus the purchase price. You may hold at most one share at a time, so you must sell your current share before buying another. All purchases and sales must take place within that working period and follow the chronological order above. You may also make no transactions, earning a profit of 0.

For each working period, independently find the maximum total profit you can earn.

Input

The first line contains three integers n, k, q ($1 \leq n \leq 10^5$, $1 \leq k \leq 800$, $1 \leq q \leq 3 \times 10^5$): the number of time slots in a day, the maximum number of transactions per working period, and the number of working periods, respectively.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \leq a_i \leq 10^6$), where a_i is the stock price in the i -th time slot of each day.

Each of the next q lines contains two integers l_i, r_i ($1 \leq l_i, r_i \leq n$), describing the i -th working period.

Let len_i be the number of time slots in this period. If $l_i \leq r_i$, then $\text{len}_i = r_i - l_i + 1$; otherwise, $\text{len}_i = n - l_i + 1 + r_i$.

For each working period, l_i and len_i are generated independently and uniformly at random from $[1, n]$ and $[\max(1, \lfloor 0.15n \rfloor), \max(1, \lfloor 0.85n \rfloor)]$, respectively. The value of r_i is uniquely determined by l_i and len_i . Specifically, when the interval is $[1, 1]$, the random result will always be 1.

There are exactly 50 test cases, excluding the sample.

Output

Output q lines. The i -th line should contain one integer: the maximum profit you can earn during the i -th working period with at most k transactions.

Example

standard input	standard output
6 2 3	7
3 1 4 1 5 9	4
2 5	0
5 2	
4 4	

Problem B. Best Trades Again

Input file: standard input
Output file: standard output
Time limit: 4 seconds
Memory limit: 512 megabytes

This is another version of Problem A, All the Trades Are the Best. In this version, each working period has its own transaction limit and lies entirely within one day. The working periods are not required to be generated randomly.

You are a trader at an investment company. One day, you discover that the stock you're responsible for trading follows an exactly repeating daily pattern: the day is divided into n equal time intervals, and the stock's price during the i -th interval is always a_i . As the n -th interval of one day concludes, the first interval of the next day immediately follows, with the price shifting from a_n to a_1 . This is a once-in-a-lifetime opportunity — if you can just buy and sell according to this pattern you've already figured out, during your working hours, there's a guaranteed way to profit!

You are given q independent working periods. The i -th period starts at slot l_i and ends at slot r_i , including both endpoints, where $l_i \leq r_i$. The slots in chronological order are $l_i, l_i + 1, \dots, r_i$.

In the i -th working period, you may complete at most k_i transactions. Each transaction consists of buying one share and selling it in a later time slot, with profit equal to the sale price minus the purchase price. You may hold at most one share at a time, so you must sell your current share before buying another. All purchases and sales must take place within that working period and follow the chronological order above. You may also make no transactions, earning a profit of 0.

For each working period, independently find the maximum total profit you can earn.

Input

The first line contains two integers n, q ($1 \leq n \leq 10^5$, $1 \leq q \leq 10^5$): the number of time slots in a day and the number of working periods, respectively.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \leq a_i \leq 10^6$), where a_i is the stock price in the i -th time slot of each day.

Each of the next q lines contains three integers l_i, r_i, k_i ($1 \leq l_i \leq r_i \leq n$, $0 \leq k_i \leq \lfloor (r_i - l_i + 1)/2 \rfloor$), describing the i -th working period and its transaction limit.

There is no randomness guarantee for the working periods or their transaction limits.

Output

Output q lines. The i -th line should contain one integer: the maximum profit you can earn during the i -th working period with at most k_i transactions.

Example

standard input	standard output
6 7	8
3 1 4 1 5 9	11
1 6 1	11
1 6 2	4
1 6 3	7
2 5 1	0
2 5 2	0
4 4 0	
1 6 0	

Note

In the first query, buy in slot 2 and sell in slot 6, earning $9 - 1 = 8$.

In the second query, buy in slot 2, sell in slot 3, buy again in slot 4, and sell in slot 6. The total profit is $(4 - 1) + (9 - 1) = 11$. Allowing a third transaction in the third query does not increase the maximum profit.

In the sixth and seventh queries, $k_i = 0$, so no transactions are allowed and the answer is 0.

Problem C. Cross the Road

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

You are about to cross the road, but an e-bike is speeding along it.

In the Cartesian plane, a road of width w extends infinitely in the x -direction and occupies the strip $0 \leq y \leq w$. You start from the fixed point $(0, 0)$ and want to reach any point on the upper boundary $y = w$, while avoiding an e-bike on the road.

The e-bike's body is represented by a line segment parallel to the x -axis. At time $t = 0$, its endpoints are (x_1, y_c) and (x_2, y_c) , where $x_1 < x_2$. The e-bike moves with constant horizontal velocity u , so at time t its endpoints are at $(x_1 + ut, y_c)$ and $(x_2 + ut, y_c)$.

You may move in any direction or wait, but your speed must never exceed v . Equivalently, for any two times $0 \leq t_1 \leq t_2$, the Euclidean distance between your positions at those times must be at most $v(t_2 - t_1)$. At no time may you be on the e-bike's segment except at its endpoints; touching either endpoint is allowed.

You start from $(0, 0)$ at time $t = 0$. Find the minimum time needed to reach any point on $y = w$ safely. It can be shown that an answer always exists.

Input

The first line contains an integer T ($1 \leq T \leq 2000$), the number of test cases.

Each test case consists of one line containing six integers w, x_1, x_2, y_c, u, v ($2 \leq w \leq 4 \times 10^4$, $-4 \times 10^4 \leq x_1 < x_2 \leq 4 \times 10^4$, $1 \leq y_c \leq w - 1$, $-4 \times 10^4 \leq u \leq 4 \times 10^4$, $1 \leq v \leq 4 \times 10^4$). Here, w is the width of the road, (x_1, y_c) and (x_2, y_c) are the e-bike's endpoints at time $t = 0$, u is the e-bike's horizontal velocity, and v is your maximum speed.

Output

For each test case, output one real number on a separate line: the minimum time needed to reach $y = w$ safely.

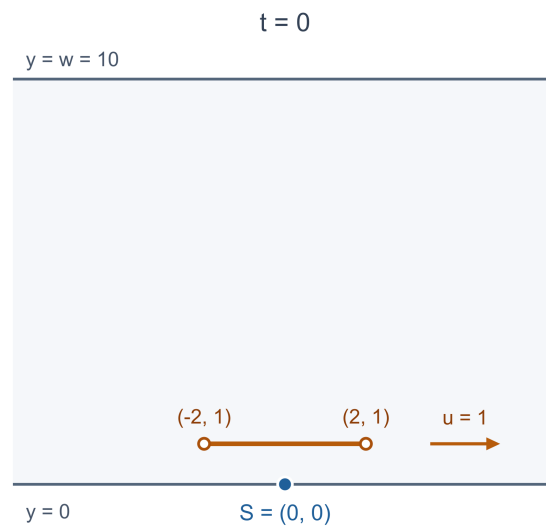
Your answer is considered correct if its absolute or relative error does not exceed 10^{-6} . More precisely, for each value a you output and the corresponding reference value b , the requirement is $\frac{|a-b|}{\max(1, |b|)} \leq 10^{-6}$.

Example

standard input	standard output
3	10.000000000
10 100 110 5 3 1	5.286299648
10 -2 2 1 1 2	10.250000000
10 -2 2 1 -1 1	

Note

The figure below shows the initial state ($t = 0$) of the second test case in the sample. The orange segment represents the e-bike, and the blue point $S = (0, 0)$ is your starting point. The arrow indicates the e-bike's direction of motion, with $u = 1$.



Problem D. Dodge the Car

Input file: **standard input**
Output file: **standard output**
Time limit: 2 seconds
Memory limit: 512 megabytes

You are about to cross the road, but a car is speeding along it.

In the Cartesian plane, a road of width w extends infinitely in the x -direction and occupies the strip $0 \leq y \leq w$. You want to cross it from the lower boundary $y = 0$ to the upper boundary $y = w$, while avoiding a car on the road.

The car's body is represented by a simple polygon with n vertices. At time $t = 0$, its vertices are $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ in counterclockwise boundary order. The car moves with constant horizontal velocity u , so at time t its i -th vertex is at $(x_i + ut, y_i)$.

You may move in any direction or wait, but your speed must never exceed v . Equivalently, for any two times $0 \leq t_1 \leq t_2$, the Euclidean distance between your positions at those times must be at most $v(t_2 - t_1)$. At no time may you be strictly inside the car's body; touching its boundary is allowed.

You are given q independent queries. In the j -th query, you start from $(s_j, 0)$ at time $t = 0$. Find the minimum time needed to reach any point on $y = w$ safely. For every query, the car starts again from the same given configuration at $t = 0$; the queries do not affect one another. It can be shown that an answer always exists.

Input

The first line contains an integer T ($1 \leq T \leq 5 \times 10^4$), the number of test cases.

Each test case begins with a line containing four integers n, w, u, v ($3 \leq n \leq 5 \times 10^5$, $1 \leq w, v \leq 10^9$, $-10^9 \leq u \leq 10^9$): the number of vertices of the car, the width of the road, the car's horizontal velocity, and your maximum speed, respectively.

The next n lines describe the car at time $t = 0$. The i -th line contains two integers x_i, y_i ($-10^9 \leq x_i \leq 10^9$, $0 \leq y_i \leq w$), the coordinates of its i -th vertex. The vertices are given in counterclockwise boundary order and form a simple polygon: non-adjacent edges have no common point, adjacent edges intersect only at their common endpoint, and no three consecutive vertices are collinear.

The next line contains an integer q ($1 \leq q \leq 5 \times 10^5$), the number of queries. Each of the following q lines contains an integer s_j ($-10^9 \leq s_j \leq 10^9$), specifying the starting point $(s_j, 0)$ for the j -th query.

It is guaranteed that both the sum of n and the sum of q over all test cases do not exceed 5×10^5 .

Output

For each query, output one real number on a separate line: the minimum time needed to reach $y = w$ safely.

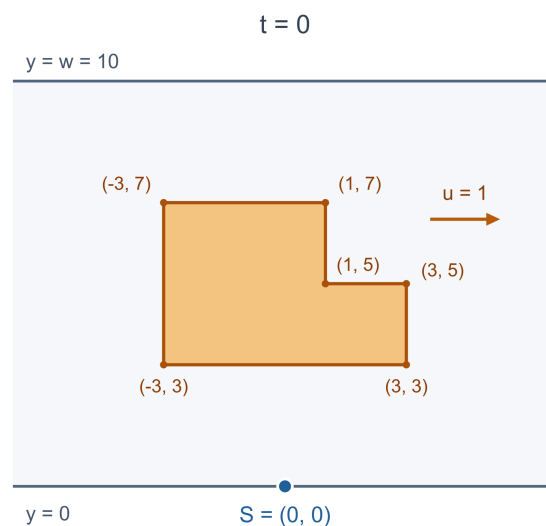
Your answer is considered correct if its absolute or relative error does not exceed 10^{-6} . More precisely, for each value a you output and the corresponding reference value b , the requirement is $\frac{|a-b|}{\max(1, |b|)} \leq 10^{-6}$.

Example

standard input	standard output
3	3.504497942940
6 10 1 3	3.333333333333
-3 3	3.333333333333
3 3	20.211324865405
3 5	20.000000000000
1 5	1.885714285714
1 7	2.412409788555
-3 7	
3	
0	
6	
-6	
4 20 5 1	
-12 2	
2 2	
2 10	
-2 10	
2	
0	
-12	
4 8 2 5	
-5 0	
5 0	
5 8	
-5 8	
2	
-3	
3	

Note

The figure below shows the initial state ($t = 0$) for the first query of the first test case in the sample. The orange polygon represents the car, and the blue point $S = (0, 0)$ is your starting point. The arrow indicates the car's direction of motion, with $u = 1$.



Problem E. EPS.AC

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

EPS.AC is the problem-setting team for this contest. Its members are Colin, Claris, and Little_Sheep_Yawn.

They have prepared n candidate problems for this contest. Each member ranks the n problems from easiest to hardest according to their own judgment. To balance the contest's difficulty and distinguish contestants of different levels, they want to divide these problems into **difficulty tiers**, ordered from easiest to hardest. Every problem must belong to exactly one tier. Problems within the same tier are not ordered by difficulty, and each tier must contain at least one problem.

Arrange the tiers from left to right, from easiest to hardest. For a boundary between two consecutive tiers, the left side contains **all problems in the tiers before it**, and the right side contains **all problems in the tiers after it**. A member supports this boundary if every problem on its left appears before every problem on its right in that member's ranking.

A division into tiers is valid if every boundary has the support of **at least two** members. Different boundaries may have different supporters.

More difficulty tiers mean better problem setting. To help them create a good contest, find the maximum possible number of tiers.

Input

The input contains multiple test cases. The first line contains an integer t ($1 \leq t \leq 2 \times 10^5$), the number of test cases.

For each test case:

- The first line contains an integer n ($1 \leq n \leq 2 \times 10^5$).
- Each of the next three lines contains a permutation of $1, 2, \dots, n$, giving the rankings of Colin, Claris, and Little_Sheep_Yawn, respectively.

It is guaranteed that the sum of n over all test cases is at most 2×10^5 .

Output

For each test case, print the maximum possible number of tiers.

Example

standard input	standard output
3	3
6	4
2 1 4 5 6 3	1
1 2 3 4 5 6	
3 1 4 2 6 5	
4	
1 3 2 4	
1 2 4 3	
2 1 3 4	
3	
1 2 3	
2 3 1	
3 1 2	

Note

In the first test case, the problems can be divided into three tiers $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, listed from easiest to hardest. Colin and Claris support the first boundary, while Claris and Little_Sheep_Yawn support the second. This division is optimal.

In the second test case, each of the four problems can form its own tier, ordered from easiest to hardest as $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$. The three boundaries are supported by Colin and Claris, by Claris and Little_Sheep_Yawn, and by Colin and Little_Sheep_Yawn, respectively.

In the third test case, no boundary is valid, so all problems must be placed in a single tier.

Problem F. Flight of the Night Butterfly

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

You need to guide a night butterfly through the darkness. Its starting point is the common center of six radial tracks, numbered clockwise from 0 to 5. Track 0 is the **main track**; the other five are **auxiliary tracks**.

Only certain intervals on the tracks are lit. The main track has one set of lit intervals, while all five auxiliary tracks share another set. All lit intervals are closed, and their endpoints represent distances from the center.

Before each flight, you may choose a **positive real** speed d , which remains fixed throughout the flight. The butterfly starts at the center at time 0. At every positive integer time t , it is on track $(t \bmod 6)$ at a distance $t \cdot d$ from the center. Thus, it visits tracks 1, 2, 3, 4, 5, 0, 1, 2, ... in this order.

You are given q queries, each specifying a target distance T . You want the butterfly to reach the main track at some time k , at a distance T from the center. Thus, k must be a positive multiple of 6, and $k \cdot d = T$. At every integer time $1, 2, \dots, k$, the butterfly must be within a lit interval on its current track, or it will fall into darkness and lose its way. There are no restrictions at other times, including time 0.

For each query, find the earliest time k at which the butterfly can safely reach its target, or print -1 if it cannot. The queries are independent, and you may choose d separately for each query.

Input

The first line contains two integers n and m ($1 \leq n, m \leq 50$), representing the numbers of lit intervals on the main track and in the set shared by all five auxiliary tracks, respectively.

Each of the next n lines contains two integers l and r ($1 \leq l \leq r \leq 10^{18}$), representing a lit interval $[l, r]$ on the main track.

Each of the next m lines contains two integers l and r ($1 \leq l \leq r \leq 10^{18}$), representing a lit interval $[l, r]$ shared by all five auxiliary tracks.

Within each set, the intervals are pairwise disjoint and are given in increasing order of their left endpoints. In particular, consecutive intervals satisfy $l_{i+1} > r_i$.

The next line contains an integer q ($1 \leq q \leq 100$), representing the number of queries.

Each of the next q lines contains an integer T ($1 \leq T \leq 10^{18}$), representing the target distance of one query.

Output

For each query, print on a separate line the earliest time k at which the butterfly can safely reach its target, or -1 if it cannot.

Example

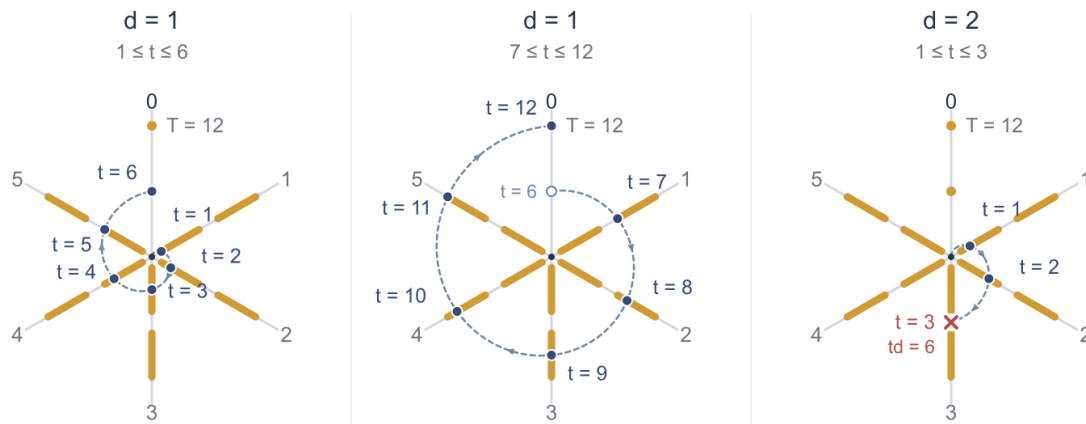
standard input	standard output
2 2	6
6 6	12
12 12	-1
1 5	
7 11	
3	
6	
12	
18	

Note

For $T = 6$, choosing $d = 1$ lets the butterfly reach the target at time $k = 6$. The distances at the first five integer times are 1, 2, 3, 4, 5, all in the lit interval $[1, 5]$ on the auxiliary tracks. At the sixth integer time, the distance is 6, which is in the lit interval $[6, 6]$ on the main track.

For $T = 12$, choosing $d = 1$ lets the butterfly reach the target at time $k = 12$. Its distances while on the auxiliary tracks are all in $[1, 5] \cup [7, 11]$, while its distances on the main track are 6 and 12. To arrive at time $k = 6$, it would need $d = 2$, but then at time 3 it would be at an unsafe position at distance 6 from the center on an auxiliary track.

The diagrams below show the query $T = 12$. The first two use $d = 1$, and the third uses $d = 2$. Gold segments and points are lit intervals. Dark blue dots are labeled with their times, and the red cross marks a position in darkness. The second panel continues from the hollow dot at time 6. Dashed arrows show only the order of integer-time positions, not the actual flight path between them.



For $T = 18$, the target is outside every lit interval on the main track, so it cannot be reached safely.

Problem G. Growing Prefix Span

Input file: **standard input**
Output file: **standard output**
Time limit: 4 seconds
Memory limit: 512 megabytes

For a permutation of $1, 2, \dots, n$, let s_i be the length of the **shortest contiguous subarray** containing all values $1, 2, \dots, i$.

Let $\text{pos}_p(x)$ be the position of x in the permutation; we have

$$s_i = \max_{1 \leq x \leq i} \text{pos}_p(x) - \min_{1 \leq x \leq i} \text{pos}_p(x) + 1.$$

For each i , an interval constraint $[l_i, r_i]$ is given.

Please count the permutations satisfying $l_i \leq s_i \leq r_i$ for every i . Print the answer modulo 998244353.

Input

The first line contains an integer n ($1 \leq n \leq 2 \cdot 10^5$).

Each of the next n lines contains two integers l_i and r_i ($1 \leq l_i \leq r_i \leq n$).

Output

Print the number of valid permutations modulo 998244353.

Please note that a valid permutation may not exist.

Example

standard input	standard output
3 1 1 2 2 3 3	4

Note

The valid permutations in the example are $(1, 2, 3)$, $(2, 1, 3)$, $(3, 1, 2)$, and $(3, 2, 1)$.

Problem H. Harmonizing Charges

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

There are n point charges at pairwise distinct positions on a number line, numbered $1, 2, \dots, n$. The i -th point charge has charge q_i and coordinate x_i . In this problem, the magnitude of the Coulomb force between two distinct point charges i and j is defined as $\frac{q_i q_j}{(x_i - x_j)^2}$.

You need to partition all the charges into two nonempty sets A and B . Some charges may already have been assigned to one of the sets, and these assignments cannot be changed.

Complete the remaining assignments so that the minimum force over all pairs of distinct charges belonging to the same set is as large as possible.

Input

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

The first line of each test case contains an integer n ($4 \leq n \leq 10^5$), the number of point charges.

Each of the next n lines contains two integers q_i, x_i and a character s_i ($1 \leq q_i \leq 15000$, $1 \leq x_i \leq 2 \times 10^5$; s_i is A, B, or ?), describing the charge, coordinate, and assignment of the i -th point charge:

- If s_i is A, the charge has already been assigned to set A .
- If s_i is B, the charge has already been assigned to set B .
- If s_i is ?, the charge has not yet been assigned to either set.

Within each test case, all coordinates are pairwise distinct, and at least one charge is unassigned.

The sum of n over all test cases does not exceed 5×10^5 .

Output

For each test case, output one line containing the maximum possible value of the minimum force between two distinct charges in the same set. Express the answer as a fraction in lowest terms, in the format p/q , where p and q are positive coprime integers.

Example

standard input	standard output
2	10/9
5	2/1
3 6 ?	
2 4 ?	
3 1 ?	
5 7 ?	
4 2 ?	
4	
1 1 A	
2 2 B	
3 3 ?	
4 4 B	

Problem I. Integer Triple

Input file: `standard input`
Output file: `standard output`
Time limit: 1 second
Memory limit: 512 megabytes

Given an integer n , find three integers a, b, c satisfying

- $|a|, |b|, |c| \leq 10^{18}$;
- $n = ab(a + b) + bc(b + c) + ac(a + c)$.

If no such triple of integers exists, report that there is no solution.

Input

The input contains one integer n ($0 \leq n \leq 10^{18}$).

Output

If there is no solution, output -1 . Otherwise, output three integers a, b, c that satisfy all the requirements.

If there are multiple valid answers, output any one of them.

Examples

standard input	standard output
3	-1
6	1 1 1

Problem J. Jobs through Pipelines

Input file: **standard input**
Output file: **standard output**
Time limit: 2 seconds
Memory limit: 512 megabytes

You need to schedule n jobs on a pipeline. The jobs form a sequence, and job i has workload a_i . The pipeline consists of m **stages** in series.

Before processing begins, you may choose an integer t ($0 \leq t \leq n$). This moves the first t jobs to the end without changing their relative order, obtaining $a_{t+1}, a_{t+2}, \dots, a_n, a_1, a_2, \dots, a_t$. Choosing $t = 0$ or $t = n$ leaves the sequence unchanged.

Next, you need to partition the resulting sequence into at most k nonempty contiguous segments, each forming a **batch**. The workload of a batch is the sum of the workloads of its jobs.

The batches then enter the pipeline in order. Each batch passes through stages $1, 2, \dots, m$ in this order, taking time equal to its workload at each stage. Each stage processes at most one batch at a time, following the batches' left-to-right order in the sequence. After finishing at one stage, a batch may wait for the next stage to become available. While it waits, it no longer occupies the previous stage, which may process the next batch. Different stages may process different batches simultaneously.

The pipeline starts processing at time 0. Choose the prefix to move and the partition into batches so that all jobs finish as early as possible.

Input

The first line contains three integers n, m, k ($1 \leq n, m \leq 5 \times 10^5$, $1 \leq k \leq n$): the number of jobs, the number of stages, and the maximum number of batches.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \leq a_i \leq 10^6$), the workloads of the jobs.

Output

Print one integer: the minimum possible time at which all jobs have finished.

Example

standard input	standard output
4 3 2 3 1 4 2	20

Note

An optimal choice is to move the prefix $[3, 1, 4]$ to the end, obtaining the sequence $[2, 3, 1, 4]$, and partition it into two batches $[2, 3]$ and $[1, 4]$. Both batches have workload 5. They can start at the first stage at times 0 and 5, and finish at the third stage at times 15 and 20, respectively. Thus, all jobs finish at time 20, which is the minimum possible time.

Problem K. Killer Die

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

You have a fair n -sided die with positive integer face values $a_1, a_2, \dots, a_n$. You want to design another fair m -sided die with positive integer face values $b_1, b_2, \dots, b_m$. Every face of a die is equally likely to be rolled. Face values may repeat on either die.

Roll each die once, independently. The new die wins if and only if its value is strictly greater than the original die's value; a tie is not a win.

Find the minimum possible sum $b_1 + b_2 + \dots + b_m$ of the new die's face values such that it wins with probability strictly greater than 50%.

Input

The first line contains two integers n, m ($2 \leq n \leq 50$, $1 \leq m \leq 10^9$), the numbers of faces of the original die and the new die, respectively.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \leq a_i \leq 10^9$), the face values of the original die.

Output

Print one integer: the minimum possible sum of the new die's face values such that its probability of winning is strictly greater than 50%.

Examples

standard input	standard output
6 6 1 2 3 4 5 6	25
3 2 1 1 2	4
4 4 3 7 10 11	29

Note

In the first example, an optimal new die has face values 1, 1, 2, 7, 7, 7, whose sum is 25. It wins $0 + 0 + 1 + 6 + 6 + 6 = 19$ of the 36 equally likely pairs of faces.

In the second example, an optimal new die has face values 2, 2. Each face beats the two faces labeled 1 on the original die, so the new die wins 4 of the 6 equally likely pairs. Its face values sum to 4.

In the third example, an optimal new die has face values 1, 4, 12, 12. It wins $0 + 1 + 4 + 4 = 9$ of the 16 equally likely pairs, and its face values sum to 29.

Problem L. Logging Compression

Input file: **standard input**
Output file: **standard output**
Time limit: 1 second
Memory limit: 512 megabytes

A system contains n types of programs, indexed from 1 to n . Once an instance of program type k starts, it generates a log entry every k seconds—specifically, at times $k, 2k, 3k, \dots$ relative to its start time.

At time 0, the system executes m commands simultaneously: the i -th command launches one instance for each program type with an index in the range $[l_i, r_i]$. Note that if the ranges of multiple commands overlap, multiple instances of the same program type may be launched, and each instance generates logs independently.

The system records all log entries generated at times $1, 2, \dots, n$. Each record contains only the index of the program type corresponding to the instance that generated the log.

Your task is to design an algorithm to compress these logs: you must assign a non-empty binary string as an **identifier** to **all the n program types**, ensuring that no identifier is a prefix of another. The cost of encoding a single record is the length of the identifier assigned to the corresponding program type, and the total cost is the sum of the costs of all records. Find the minimum possible total cost.

Input

The first line contains two integers n, m ($1 \leq n \leq 10^{10}$, $1 \leq m \leq 10^5$), representing the number of types of programs and the number of commands.

Each of the next m lines contains two integers l_i, r_i ($1 \leq l_i \leq r_i \leq n$), representing a start command.

Output

Print a single integer: the minimum total cost.

Examples

standard input	standard output
5 1 1 5	20
6 2 1 3 4 6	32

Note

Assume that logs are recorded in chronological order; for logs generated at the same time, the records follow a monotonically non-decreasing order based on the index of the corresponding program types.

For the first sample, the logs are $[1, 1, 2, 1, 3, 1, 2, 4, 1, 5]$. An optimal assignment of identifiers is (program 1: “0”, program 2: “100”, program 3: “101”, program 4: “110”, program 5: “111”). The cost is $5 \times 1 + 2 \times 3 + 1 \times 3 + 1 \times 3 + 1 \times 3 = 20$.

For the second sample, the logs are $[1, 1, 2, 1, 3, 1, 2, 4, 1, 5, 1, 2, 3, 6]$. An optimal assignment of identifiers is (program 1: “0”, program 2: “10”, program 3: “1100”, program 4: “1101”, program 5: “1110”, program 6: “1111”). The cost is $6 \times 1 + 3 \times 2 + 2 \times 4 + 1 \times 4 + 1 \times 4 + 1 \times 4 = 32$.

Problem M. Median Equals X

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 512 megabytes

You are given an integer sequence $a_1, a_2, \dots, a_n$ of length n , and an integer x .

Define a **subsequence** of the sequence as follows: for any non-empty set of indices $\{i_1, i_2, \dots, i_m\}$ ($i_1 < i_2 < \dots < i_m$) chosen from $1, \dots, n$, taking $a_{i_1}, \dots, a_{i_m}$ in their original order is a subsequence. Define its **median** as follows: sort the m chosen values into $v_1 \leq v_2 \leq \dots \leq v_m$, then:

- if m is odd, the median is the middle value $v_{(m+1)/2}$;
- if m is even, the median is the **average** of the two middle values, $\frac{v_{m/2} + v_{m/2+1}}{2}$ (this average need not be an integer, but the problem only cares whether it is **exactly equal** to x).

Find the number of subsequences whose median is exactly x . Since this number can be very large, output it modulo 998244353.

Note: even if two different index sets happen to pick out the exact same multiset of values (possible since the sequence may contain repeated numbers), they are still counted as two distinct subsequences. It is the index set itself, not the multiset of chosen values, that distinguishes subsequences. The empty index set is not a subsequence and is never counted.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$).

For each testcase:

- The first line contains two integers n and x ($1 \leq n \leq 2 \times 10^5$, $-10^9 \leq x \leq 10^9$), representing the length of the sequence a and the target median value.
- The second line contains n integers $a_1, a_2, \dots, a_n$ ($-10^9 \leq a_i \leq 10^9$), representing the elements of the given sequence.

It is guaranteed that the sum of n over all test cases does not exceed 2×10^5 .

Output

For each test case, output one line with one integer: the number of subsequences whose median is exactly x , modulo 998244353.

Example

standard input	standard output
2	14
5 3	1
1 3 5 3 2	
2 0	
-1 1	