Association for
Computing Machinery
Hong Kong Chapter

ACM-HK Programming Contest 2026

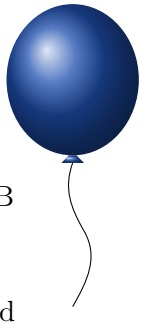
Problem Set

Hosted by ACM Hong Kong Chapter and The University of Hong Kong.

Sponsors and Supporting Organizations

BLANK PAGE

A Apple and Cinnamon



Time limit: 1.0s
Memory limit: 1024MB

Alice is making a signature dessert that relies on a delicate harmony of two ingredients: apple and cinnamon. To achieve this, Alice reaches into a large, opaque canvas bag filled with flavor drops.

We know for certain that the bag contains at least X apple drops and at least Y cinnamon drops. However, there may be more drops of either flavor in the bag.

Among all possible final numbers of apple and cinnamon drops satisfying these lower bounds, Alice draws exactly two drops uniformly at random, without replacement. Alice's greatest hope is to draw one of each flavor, allowing apple and cinnamon to meet in a single trial dessert. Therefore, Alice wants to know the minimum possible probability of drawing two drops of the same flavor.

Input

The only line of the input contains two integers X and Y ($1 \le X, Y \le 10^9$) — the minimum required number of apple drops and cinnamon drops in the bag, respectively.

Output

Print one real number — the minimum possible probability that the two drawn drops have the same flavor.

Your answer will be considered correct if its absolute or relative error does not exceed 10^{-9}.

Samples

Sample input 1	Sample output 1
3 5	0.44444444444444444

Explanation of sample 1.
If the bag contains a apple drops and b cinnamon drops, the probability of drawing two drops of the same flavor is

$$\frac{a(a-1)+b(b-1)}{(a+b)(a+b-1)}.$$

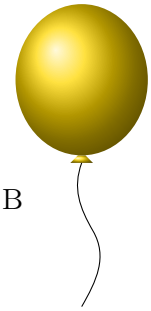
For $X = 3$ and $Y = 5$, one optimal choice is $(a, b) = (4, 5)$, giving $\frac{4\cdot3+5\cdot4}{9\cdot8} = \frac{4}{9}$.

Sample input 2	Sample output 2
1 1	0.00000000000000000000

Explanation of sample 2.
With $X = Y = 1$, Alice can use exactly one drop of each flavor. Then every draw of two drops contains one of each flavor, so the probability of drawing two equal flavors is 0.

Sample input 3	Sample output 3
3971 1368	0.49993703563782898879

B Don't Look Back in Anger



Time limit: 1.0s
Memory limit: 1024MB

Alice possesses k precious memories, each assigned a unique happiness value from 1 to k corresponding to the chronological order in which they occurred. To explore her past, she has inscribed a sequence of n magical spells. The i-th spell rearranges her memories according to a permutation A_i. When a continuous sequence of spells is cast, their effects compound, creating a new, complex rearrangement of her mind.

As spells shuffle the timeline of her memories, a later memory (one with a higher happiness value) might end up placed before an earlier memory. This temporal dissonance creates what Alice calls a regretful pair—a moment where the natural chronological order of happiness has been inverted. Driven by curiosity, she wishes to measure the total emotional dissonance by counting every regretful pair generated across all possible contiguous segments of spells.

Formally, you are given k memories and a sequence of n permutations, where the i-th permutation is denoted as A_i. Each A_i is a permutation of $\{1, 2, \ldots, k\}$. For any two permutations p and q, their composition $p \circ q$ is defined as

$$(p \circ q)(x) = p(q(x)).$$

For any contiguous segment of spells $[l, r]$ (where $1 \le l \le r \le n$), let $P_{l,r}$ represent the final permutation after the spells are compounded. It is defined as:[1]

$$P_{l,r} = A_l \circ A_{l+1} \circ \cdots \circ A_r.$$

Your task is to compute the total number of regretful pairs across all contiguous segments of spells. Formally, calculate:

$$\sum_{1 \le l \le r \le n} \mathrm{inv}(P_{l,r}),$$

where $\mathrm{inv}(p)$ denotes the number of inversions in permutation p, formally defined as the number of pairs (x, y) such that $1 \le x < y \le k$ and $p(x) > p(y)$.

Input

The first line contains two integers n and k ($1 \le n, k \le 10^5$, $nk \le 10^5$).

Each of the next n lines contains k integers. The i-th of these lines contains the permutation $A_i(1), A_i(2), \ldots, A_i(k)$.

Output

Print one integer on a line, denoting the answer.

[1] By associativity of permutation composition, the parenthesization of this product does not matter. For example, $(A_l \circ A_{l+1}) \circ A_{l+2} = A_l \circ (A_{l+1} \circ A_{l+2})$. The order of the permutations is fixed as written.

Samples

Sample input 1	Sample output 1
2 3 1 3 2 2 3 1	6

Explanation of sample 1.

- The inversion counts of $A_1 = [1, 3, 2]$ and $A_2 = [2, 3, 1]$ are 1 and 2.
- Their composition is $A_1 \circ A_2 = [3, 2, 1]$, whose inversion count is 3.
- Summing over the three non-empty contiguous segments gives $1 + 2 + 3 = 6$.

Sample input 2	Sample output 2
6 5 5 3 1 4 2 2 4 3 1 5 5 4 2 3 1 1 3 4 5 2 2 5 3 4 1 1 2 5 4 3	116

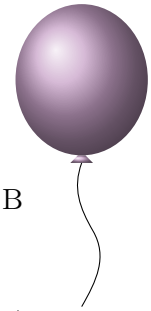
Explanation of sample 2.

There are 21 contiguous segments. Grouped by segment length, the sums of inversion counts are

$$32,\ 28,\ 24,\ 12,\ 12,\ 8.$$

Their total is 116.

C Game: Adversarial Distance Oracle



Time limit: 2.0s
Memory limit: 1024MB

Alice and Bob are playing a game on a tree T with n vertices. Alice wants to identify a secret vertex x. However, Bob is adversarial and does not have to choose x in advance.

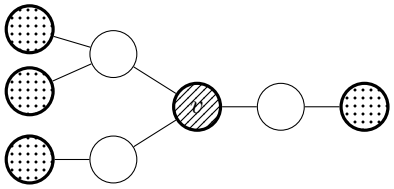
Initially, every vertex of T is considered a "possible" candidate for x. In each turn, Alice chooses a vertex v and asks for the distance from v to x. Bob responds with a non-negative integer d. Alice then removes every candidate vertex u such that $\text{dist}(u, v) \neq d$.

Bob's answer must be consistent with at least one remaining candidate. In other words, after Alice removes all vertices u with $\text{dist}(u, v) \neq d$, the candidate set must still be non-empty. Subject to this rule, Bob chooses his answers to force Alice to use as many queries as possible.

Alice wins as soon as exactly one candidate vertex remains. Alice chooses her queries optimally to minimize the number of queries.

Given the structure of the tree, find the minimum number of queries Alice needs to guarantee a win, regardless of Bob's strategy.

For example, if Alice queries a vertex v and Bob answers $d = 2$, then all vertices at distance exactly 2 from v remain possible, and all other vertices are removed.



Querying v and receiving answer 2: the hatched vertex is queried, dotted vertices remain possible, and plain vertices are removed.

Input

The first line contains one integer t ($1 \leq t \leq 10^5$), the number of test cases.

Each test case begins with one integer n ($2 \leq n \leq 2000$), the number of vertices.

Each of the next $n - 1$ lines contains two integers a_i and b_i ($1 \leq a_i, b_i \leq n$, $a_i \neq b_i$), denoting an edge of the tree.

It is guaranteed that the edges of each test case form a tree and that $\sum n^2 \leq 2000^2$ over all test cases.

Output

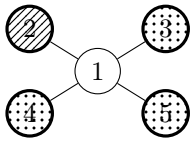
For each test case, print one integer: the minimum number of queries Alice needs in the worst case.

Samples

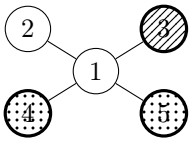
Sample input 1	Sample output 1
2 4 1 2 2 3 3 4 5 1 2 1 3 1 4 1 5	1 3

Explanation of sample 1.

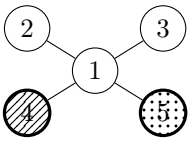
- In the first test case, the tree is a path on four vertices. Querying vertex 1 gives possible distances $0, 1, 2, 3$, all distinct, so one query is enough.
- In the second test case, the tree is a star centered at vertex 1:
- In the following diagrams, the hatched vertex is the queried vertex, dotted vertices remain possible after Bob's answer, and plain vertices have been removed.



Query 2, answer 2: vertices $3, 4, 5$ remain possible.



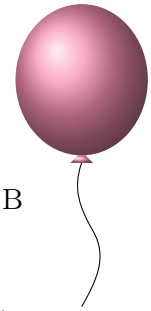
Query 3, answer 2: vertices $4, 5$ remain possible.



Query 4, answer 2: only vertex 5 remains.

- This shows that three queries are sufficient. Alice first queries leaf 2. If Bob answers 0 or 1, the secret is immediately determined; the worst answer is 2, leaving leaves $3, 4, 5$. Then Alice queries leaf 3, and in the worst case leaves $4, 5$ remain. One final query distinguishes them.
- Two queries are not enough. After the first query, Bob can keep at least three leaves possible. Among three leaves of a star, one more distance query cannot distinguish all of them, because at least two of the leaves have the same distance to the queried vertex.

D Game: Coin Flip



Time limit: 2.0s
Memory limit: 1024MB

Alice and Bob play a sequence of games with a biased coin. The coin lands heads with probability p, and tails with probability $1 - p$.

In a single game, the players toss the coin repeatedly. After each toss, suppose that the current game has lasted exactly m tosses. The game ends immediately if one of the following conditions is satisfied.

If there exists an integer $i \geq 1$ such that $2^i \mid m$, and the last 2^i tosses of the current game are

$$\underbrace{\mathrm{HH}\ldots\mathrm{H}}_{2^{i-1}}\underbrace{\mathrm{TT}\ldots\mathrm{T}}_{2^{i-1}},$$

then Alice wins the game.

If there exists an integer $i \geq 1$ such that $2^i \mid m$, and the last 2^i tosses of the current game are

$$\underbrace{\mathrm{TT}\ldots\mathrm{T}}_{2^{i-1}}\underbrace{\mathrm{HH}\ldots\mathrm{H}}_{2^{i-1}},$$

then Bob wins the game.

As soon as a game ends, the next game starts with the next toss.

Little Z recorded the first n tosses, but some characters in the record were lost and are written as `?`. Each `?` is independently equal to H with probability p, and equal to T with probability $1 - p$. The characters H and T in the record are fixed.

Given n, p, and the recorded string, compute the expected number of games won by Alice and the expected number of games won by Bob among the games that end within the first n tosses.

Input

The first line contains an integer n and a real number p ($1 \leq n \leq 200000$, $0 < p < 1$). The number p is given with exactly six digits after the decimal point.

The second line contains a string s of length n. Each character of s is either H, T, or `?`.

Output

Print two real numbers: the expected number of games won by Alice and the expected number of games won by Bob.

Your answer will be accepted if both numbers have an absolute or relative error of at most 10^{-6}.

Samples

Sample input 1	Sample output 1
8 0.400000 ??HHTTHH	0.720000000000000 1.120000000000000

Explanation of sample 1.
Only the first two tosses are unknown.

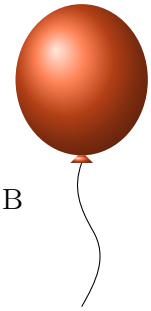
- The four completed records are HHHHTTHH, HTHHTTHH, THHHTTHH, and TTHHTTHH, with probabilities $0.16, 0.24, 0.24, 0.36$.
- Their Alice/Bob win counts are $(0, 1)$, $(2, 0)$, $(1, 1)$, and $(0, 2)$.
- Taking the weighted sum gives $(0.72, 1.12)$, matching the sample output.

Sample input 2	Sample output 2
20 0.314159 ???H???T??T?????H???	2.590680729436823 2.652863744188335

Explanation of sample 2.
This record has 16 unknown tosses.

- A completion with h heads among the unknown positions has probability $0.314159^h(1 - 0.314159)^{16-h}$.
- Summing the Alice and Bob win counts over all completions gives the two expectations printed in the sample output.

E Game: Cursor Minimum



Time limit: 2.0s
Memory limit: 1024MB

This is an interactive problem.

A permutation p of length $n = 30000$ is hidden by the judge. Your task is to find the index of the minimum value of this permutation.

The judge is non-adaptive: the permutation is chosen before the interaction starts and never changes.

Unlike ordinary comparison queries, the judge only compares your current query with your previous query. More precisely, if your previous queried index was i and now you query index j, the judge tells you the comparison between p_i and p_j.

You may make at most $q_{\max} = 42000$ queries.

The official judging data consists of 100 separate test files. Your program is run independently on each file.

Interaction

At the beginning of the interaction, your program must read two integers

$$n \quad q_{\max}.$$

For official tests, $n = 30000$ and $q_{\max} = 42000$.

To make a query, print one line in the following format:

```
? j
```

where $1 \le j \le n$.

The judge replies with one character:

- `<` if $p_i < p_j$ where i denotes the previous query (if applicable);
- `=` if this is the first query, or if $p_i = p_j$;
- `>` if $p_i > p_j$ where i denotes the previous query (if applicable).

Since p is a permutation, the reply `=` after the first query can only happen if you query the same index as in the previous query. If the judge replies with `-1`, your program must terminate immediately. This means that your program has violated the interaction protocol and will receive Wrong Answer.

To give your final answer, print one line in the following format:

```
! a
```

where a is the index that you claim contains the minimum value. After printing the answer, your program must terminate. **The final answer is not counted as a query.**

If your program makes more than $q_{\max}$ queries, queries an invalid index, prints an invalid command, or outputs an incorrect final answer, it receives Wrong Answer.

After every printed query or answer, flush the output. For example, in C++ you may use `endl` or `cout.flush()`.

Samples

Sample input 1	Sample output 1
4 5 (receiving participant's output) = (receiving participant's output) > (receiving participant's output) < (receiving participant's output)	(receiving jury's output) ? 1 (receiving jury's output) ? 2 (receiving jury's output) ? 4 (receiving jury's output) ! 2

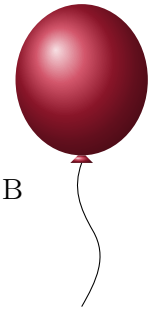
Explanation of sample 1.
This sample is only an example interaction and will not appear in the real test data. The lines of the form `(receiving ... output)` are placeholders used only to align queries and judge replies in the sample. In a real test case, the jury will not output these placeholder lines, and the participant should neither read nor print them.

In this example interaction, the hidden permutation is $p = (3, 1, 4, 2)$. The bullet points below describe the interaction shown in the sample.

- The judge first sends $n = 4$ and query limit 5.
- The first query `? 1` receives `=` because there is no previous queried index.
- The query `? 2` compares $p_1 = 3$ with $p_2 = 1$, so the judge replies `>`.
- The query `? 4` compares $p_2 = 1$ with $p_4 = 2$, so the judge replies `<`. The minimum is at index 2, so `! 2` is correct.

Local interaction tool: The attached `attachments/local_interactive.py` can reproduce this local setting with `--perm 3,1,4,2 --limit 5`, but passing it does not guarantee passing the official judging data.

F Game: Pattern Chase



Time limit: 1.0s
Memory limit: 1024MB

This is an interactive problem.

Alice and Bob are playing a game with a binary string s and a fixed integer k. Initially, there is an empty string t. The players take turns appending a character `0` or `1` to the end of t, starting with Alice.

The interaction always continues until exactly k characters have been appended to t. Alice wins if and only if the final string t contains s as a contiguous substring. Otherwise, Bob wins.

You may choose to play as either Alice or Bob. Your goal is to win the game against the jury.

Interaction

Each test run contains multiple test cases. You should first read a line with an integer T ($1 \le T \le 100$), representing the number of test cases.

For each test case, you begin the interaction by reading a binary string s and an integer k in a single line ($1 \le |s| \le k \le 100$), denoting the number of rounds and parameters of the game. Afterwards, output one word: `Alice` if you choose to play as Alice, or `Bob` if you choose to play as Bob. After that, the game starts from the empty string. Alice makes the first move. Whenever it is your turn, output one character, either `0` or `1`. Whenever it is the jury's turn, read one character, either `0` or `1`. **The game ends when the current string has length k.**

If there are multiple choices of role or moves that can make you win, you may output any of them, as long as your interaction follows the protocol and wins the game.

After every output operation you must flush the output buffer. For example, in C++ you may use `cout << value << endl;` or `cout.flush();`.

If you output an invalid token, make a move after the game has ended, fail to flush, or lose the game, you will receive Wrong Answer.

Samples

Sample input 1	Sample output 1
1 01 3 (receiving participant's output) (receiving participant's output) 0 (receiving participant's output)	(receiving jury's output) (receiving jury's output) Alice 0 (receiving jury's output) 1

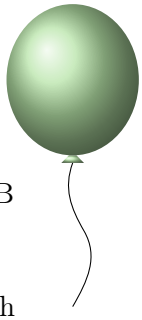
Explanation of sample 1.
The lines of the form `(receiving ... output)` are placeholders used only to align the judge stream and the contestant stream in the sample. In a real test case, the jury will not output these placeholder lines, and the participant should neither read nor print them.

The bullet points below describe the interaction shown in the sample.

- The jury sends `1`, so there is one test case, then sends `01 3`, so $s = 01$ and $k = 3$.
- The contestant chooses `Alice` and appends `0`, so the current string becomes `0`.
- The jury, playing Bob, appends `0`; now the current string is `00`.
- Alice appends `1`. The final string is `001`, which contains `01`, so Alice wins.

Local interaction tool: The attached `attachments/local_interactive.py` can be used for local testing with random jury moves, but passing it does not guarantee passing the official judging data.

G November Rain



Time limit: 1.0s
Memory limit: 1024MB

You are conducting a grand, evolving symphony. The ensemble consists of musicians, where each musician plays a specific harmonic frequency (represented by a non-negative integer). Initially, the stage is completely empty. Over n sequential steps, exactly one action is taken to change the arrangement. For each step $i = 1, 2, \ldots, n$, an operation is performed:

- If the operation is `+` (Enter): A new musician joins the ensemble. You must decide the exact frequency b_i they will play.
- If the operation is `-` (Exit): A musician leaves the stage. You must choose the frequency b_i of a musician currently performing and have exactly one of them stop playing.

At every step, the performance is anchored by the "Phantom Note." Because of the unique acoustics of the symphony, the Phantom Note is never actually played by anyone on stage. Instead, its pitch is always determined by the *lowest frequency that is currently missing from the performance.*[2]

After the i-th action, it is required that the Phantom Note must resonate at exactly a_i.

Your task is to determine whether a valid sequence of chosen frequencies $b_1, b_2, \ldots, b_n$ exists that perfectly orchestrates the required Phantom Note progression at every step, and to construct one such sequence if it does.

Input

This problem has multiple test cases. The first line of input contains a single integer t ($1 \le t \le 3 \cdot 10^5$), representing the number of test cases.

For each test case, each performance is described over three lines:

- The first line contains a single integer n ($1 \le n \le 5000$), representing the total number of sequential steps in the performance.
- The second line contains a string op of length n, consisting exclusively of the characters `+` and `-`. The character op_i dictates the nature of the i-th action: `+` signifies a musician Entering, and `-` signifies a musician Exiting.
- The third line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le n$), representing the exact required pitch of the Phantom Note after the i-th action.

It is strictly guaranteed that the sum of n^2 over all performances does not exceed 5000^2.

Output

For each performance, output your result as follows:

[2]The pitch of the Phantom Note is mathematically defined as the `mex` (minimum excluded value). Let S be a multiset of non-negative integers representing the collection of frequencies currently being played by the ensemble. The minimum excluded value, denoted as $\text{mex}(S)$, is the smallest non-negative integer x such that $x \notin S$.

If it is impossible to orchestrate the required Phantom Note progression, print a single line containing the word `NO`.

Otherwise, print two lines:

- The first line must contain the word `YES`;
- The second line must contain n non-negative integers $b_1, b_2, \ldots, b_n$, representing the specific frequency played by the entering or exiting musician at each corresponding step.

Each frequency b_i must perfectly satisfy the acoustic constraints and operational rules described in the problem statement. You are permitted to output any valid non-negative integer, provided it is representable by a standard signed 64-bit integer.

If there are multiple valid sequences of frequencies that satisfy the performance, you may print any one of them.

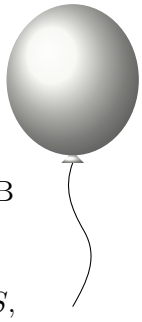
Samples

Sample input 1	Sample output 1
4 2 ++ 0 2 3 +++ 0 1 3 3 +-+ 1 0 2 6 ++-+-+ 0 2 0 0 0 1	YES 1 0 YES 2 0 1 NO YES 1 0 0 7 1 0

Explanation of sample 1.
There are four test cases in sample 1.

- In the first test case, inserting 1 keeps the mex (Phantom Note) 0, then inserting 0 makes the mex become 2.
- In the second test case, inserting $2, 0, 1$ makes the mex sequence $0, 1, 3$.
- In the third test case, mex 1 after the first operation forces insertion of 0; after erasing it, one more insertion cannot make mex 2, so the answer is `NO`.
- In the fourth test case, the printed values make the multiset evolve with mex sequence $0, 2, 0, 0, 0, 1$.

H The Real Folk Blues



Time limit: 1.0s
Memory limit: 1024MB

You are slicing into a Red Dragon Syndicate data terminal. The terminal's archive, denoted as S, initially contains n encrypted signal frequencies, each represented as a binary string of length k. These initial frequencies are indexed from 1 to n in the exact order they are extracted from the terminal's memory. To breach the core, you must synthesize new frequencies and inject them into the archive. Each synthesis operation appends exactly one new binary string to S, automatically assigning it the next available integer index.

You are equipped with two operations:

- **Phase Inversion**: Select an existing frequency s and append its exact *bitwise complement*[3];
- **Signal Triangulation**: Select three existing frequencies u, v, and w (not necessarily distinct) and append their *bitwise majority*, denoted as $\text{maj}(u, v, w)$. For every bit position i, the operation evaluates as:
$$\text{maj}(u, v, w)_i = \text{maj}(u_i, v_i, w_i).$$
For individual bits a, b, c, $\text{maj}(a, b, c)$ is 1 if at least two of them are 1, and 0 otherwise.

Your objective is to figure out if it is possible to forge a specific target bounty code t of length k. If it is possible, you need to provide a sequence of operations (at most 10^5) that successfully builds it.

Input

The first line of the terminal feed contains two integers n and k ($1 \le n, k \le 200$), representing the number of initial codes and the length of each code.

Each of the following n lines contains a binary string (`0`s and `1`s) of length k, showing a code currently in the archive.

The final line contains a single binary string t of length k, representing the target bounty code you need to forge.

Output

If it is impossible to forge the target code t, print a single line containing `NO`.

Otherwise, print `YES`. On the next line, print an integer m ($0 \le m \le 10^5$), representing the total number of operations you will use. Then, print m lines describing your operations in order:

- `1 x`: Pick the existing code at index x and apply Phase Inversion (append bitwise complement of x);
- `2 x y z`: Pick the existing codes at indices x, y, and z and apply Signal Triangulation (append maj of the existing strings with indices x, y, and z).

[3]Let s be a binary string of length k, such that $s = s_1 s_2 \dots s_k$ where each bit $s_i \in \{0, 1\}$. The bitwise complement of s, often denoted mathematically as $\neg s$, is defined as a new binary string of length k where the value of the i-th bit is exactly $1 - s_i$.

Every index you use must already exist in the archive at the moment you use it. After all m operations, at least one code in the archive must perfectly match your target t. If the target code t is already in the starting archive, you can just output $m = 0$.

If there are multiple correct ways to forge the code, you may print any valid sequence of operations.

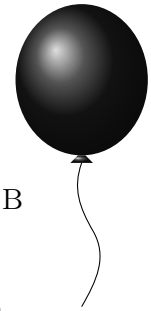
Samples

Sample input 1	Sample output 1
3 4 1000 0100 0010 1111	YES 4 1 1 1 2 1 3 2 4 5 6

Explanation of sample 1.

- The first three operations append the complements of the initial strings, creating `0111`, `1011`, and `1101`.
- The final operation takes the bitwise majority of these three strings.
- In every position at least two of them have bit 1, so the appended string is `1111`, which is the target.

I The Space between Two Worlds



Time limit: 2.0s
Memory limit: 1024MB

Alice and Bob reside in separate worlds. No matter how deeply they yearn to communicate, the universe answers only with an indifferent silence. Between their worlds lies a vast, quiet border where n ancient gates stand in a row. These gates have lain dormant for millennia until one day, perhaps moved by Alice and Bob's wishes, they begin to awaken.

The i-th gate awakens on day p_i. No two gates awaken on the same day; thus, the sequence $p_1, p_2, \ldots, p_n$ is a permutation of $\{1, 2, \ldots, n\}$.

A single awakened gate is merely a lonely fracture in reality. It might carry a fleeting whisper, but it cannot bridge the void. For a contiguous segment of gates to form a true path for Alice and Bob, at least two gates within it must be awakened. Only then does the space between the worlds stabilize enough for their messages to safely cross.

For any gate indices $i < j$, let $\mathrm{sec}(i, j)$ define the exact day the contiguous segment of gates from i to j first contains at least two awakened gates. Equivalently, $\mathrm{sec}(i, j)$ is the second smallest value among $p_i, p_{i+1}, \ldots, p_j$.

You are given q queries. Each query is defined by an interval $[L, R]$. For each query, calculate the sum of the exact days on which every continuous subsegment of length at least two fully contained within $[L, R]$ first becomes able to carry their messages:

$$\sum_{L \le i < j \le R} \mathrm{sec}(i, j).$$

If $L = R$, there is no segment containing at least two gates, so the answer is 0.

Input

The first line contains two integers n and q ($1 \le n, q \le 2 \cdot 10^5$).

The second line contains n integers $p_1, p_2, \ldots, p_n$ forming a permutation of $\{1, 2, \ldots, n\}$.

Each of the next q lines contains two integers L and R ($1 \le L \le R \le n$), describing a query.

Output

Print q lines. The t-th line must contain the answer to the t-th query.

Samples

Sample input 1	Sample output 1
4 4 3 1 4 2 1 4 2 4 1 2 3 3	18 10 3 0

Explanation of sample 1.

- For query $[1, 4]$, the second minimum values over all subarrays of length at least two are $3, 3, 2, 4, 2, 4$, summing to 18.
- For query $[2, 4]$, the values are $4, 2, 4$, summing to 10.
- For query $[1, 2]$, the only valid subarray has second minimum 3.
- For query $[3, 3]$, there is no subarray of length at least two, so the answer is 0.