A. AND-OR closure

Time limit: 1 second
Memory limit: 256 megabytes

Given a set A of n non-negative integers, we define its **AND-OR closure** as the B with smallest size such that:

- $A \subseteq B$
- If $x, y \in B$, then $(x \texttt{ AND } y) \in B$
- If $x, y \in B$, then $(x \texttt{ OR } y) \in B$

Find the size of the AND-OR closure of A.

Here `AND` denotes the bitwise AND operation, and `OR` denotes the bitwise OR operation.

Input

The first line of the input contains a single integer n $(1 \le n \le 2 \cdot 10^5)$ — the size of the set A.

The second line contains n distinct integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i < 2^{40})$ — which represent the elements of A.

Output

Print the size of the AND-OR closure of A.

Examples

standard input	standard output
4 0 1 3 5	5
5 0 1 2 3 4	8

Note

In the first sample, the AND-OR closure of A is $\{0, 1, 3, 5, 7\}$.

In the second sample, the AND-OR closure of A is $\{0, 1, 2, 3, 4, 5, 6, 7\}$.

B. Build Permutation

Time limit: 1 second
Memory limit: 256 megabytes

Given an integer array a of length n, find a permutation π of $\{1, 2, \ldots, n\}$ such that $a_i + a_{\pi_i} = a_j + a_{\pi_j}$ for all $i, j \in \{1, 2, \ldots, n\}$ or report none such permutation exists.

Input

The first line of input contains a single integer n $(1 \leq n \leq 2 \cdot 10^5)$ — the length of the array.

The second line of input contains n integers a_1, a_2, …, a_n $(0 \leq a_i \leq 10^9)$ — elements of the array.

Output

If no such permutation exists, print -1.

Otherwise, on the first line print $\pi_1, \pi_2, \ldots, \pi_n$. If several such permutations exist, you can output any of them.

Examples

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

Note

In the first sample, we have:

- $a_1 + a_{\pi_1} = a_1 + a_2 = 4 + 2 = 6$
- $a_2 + a_{\pi_2} = a_2 + a_1 = 2 + 4 = 6$
- $a_3 + a_{\pi_3} = a_3 + a_4 = 5 + 1 = 6$
- $a_4 + a_{\pi_4} = a_4 + a_3 = 1 + 5 = 6$
- $a_5 + a_{\pi_5} = a_5 + a_5 = 3 + 3 = 6$

In the second sample, no such permutation exists.

C. Christmas Sky

Time limit: 2 seconds
Memory limit: 256 megabytes

This Christmas you are going to pay a visit to your beloved grandparents. You begin to remind yourself of the beautiful memories you have with them as a child. In particular, you remember that you and your grandfather used to look at the sky every Christmas night, gazing at the bright stars, and taking photos of them.

You want to give your grandpa a special greeting card for Christmas this year, in honor of your memories together. You took a photo of last night's sky, where n bright stars (described by points in the 2D plane) can be seen. However, in the meantime, you found a very similar old photo taken some years ago, having m bright stars. While deciding for a message for your grandpa, you begin to think: how actually similar is the sky in this photo to the one on the old photo?

You try to move the new photo around (without rotating it), trying to make it overlap the most with the old photo. The photos overlap the most when the **maximum distance** (in Euclidean norm) between one of the m stars in the old photo and one of the n stars in the new photo is **as small as possible**.

Write a program that computes the minimum such distance, as well as how you should translate the new photo in order to obtain this distance.

Input

The first line of the input contains a single integer n $(1 \leq n \leq 1000)$ — the number of stars in the new photo.

Then i-th of the next n lines contains two integers x_i, y_i $(1 \leq x_i, y_i \leq 10^6)$, describing the star with coordinates (x_i, y_i) on the new photo.

The next line of the input contains a single integer m $(1 \leq m \leq 1000)$ — the number of stars in the old photo.

Then m lines follow, describing each star in the old photo in an identical manner.

All n stars in the first photo are at distinct locations. All m stars in the second photo are also at distinct locations.

Output

Output three numbers: d, t_x, and t_y, separated by space, representing the minimum distance required, as well as how much the new photo should be translated in order to obtain this minimum distance. The answer would be considered correct if your answer d, as well as the distance given by the translation vector (t_x, t_y) are within absolute or relative error of at most 10^{-7} to the real answer.

Examples

standard input	standard output

3
1 5
2 4
6 8
2
1 3
1 6
	4.03112887415 -3 -1.5

1
5 1
1
4 9
	0 -1 8

Note

In the first sample test, after translating the new photo by the vector $t = (-3, -1.5)$, we obtain the new stars at coordinates $(-2, 3.5)$, $(-1, 2.5)$ and $(3, 6.5)$. The maximum distance in this case is $\sqrt{16.25}$, given by the distance between the second star in the new photo and the second star in the old photo. It can be proved that no smaller distance may be achieved.

D. Distinct Game

Time limit: 2 seconds
Memory limit: 256 megabytes

Given two arrays $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ such that $n+m = 2k$. All values of both arrays are from 1 to k. Every value from 1 to k appears exactly twice in arrays, both occurrences could be either in the same array or in different ones.

Two players play a game. In each move, the player can take the last element of either array. The game ends when all elements are taken. The second player wins if all his elements are distinct, otherwise, the first player wins. Determine which player has a winning strategy.

Input

The first line contains a single integer t $(1 \le t \le 10^5)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains two integers n and m $(1 \le n, m \le 5 \cdot 10^5)$ — the length of the arrays.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le \frac{n+m}{2})$ — elements of the first array.

The second line of each test case contains m integers $b_1, b_2, \ldots, b_m$ $(1 \le b_i \le \frac{n+m}{2})$ — elements of the second array.

Each value from 1 to $\frac{n+m}{2}$ appears exactly twice in the arrays.

It is guaranteed that the sum of $n + m$ over all test cases does not exceed 10^6.

Output

For each test case, print 1 if the first player wins and 2 otherwise.

Example

standard input	standard output

```
4
1 3
1
1 2 2
3 3
1 2 3
2 3 1
2 4
3 1
2 3 1 2
2 2
1 2
1 2
```
	```
	2
	1
	2
	2
	```

E. Eliminate Tree

Time limit: 1 second
Memory limit: 256 megabytes

You are given a tree on n nodes. You can perform the following operations:

- Add a new vertex to the tree and connect it to exactly one already existing node
- Take an edge (u, v) with $\deg(u) = 1$ and $\deg(v) \leq 2$ and remove both of these vertices, as well as all edges connected to them, from the tree

What is the smallest number of operations you need to perform to remove all the vertices from the tree?

Input

The first line of input contains n $(1 \leq n \leq 2 \cdot 10^5)$ — the number of nodes of the tree.

The i-th of the next $n-1$ lines contains two integers u_i and v_i $(1 \leq u_i, v_i \leq n, u_i \neq v_i)$, denoting an edge between nodes u_i, v_i. It is guaranteed that these edges form a tree.

Output

Output a single integer — the smallest number of operations you need to perform to remove all the vertices from the tree.

Examples

standard input	standard output

5
1 2
2 3
3 4
3 5
	4

4
1 2
2 3
3 4
	2

Note

In the first sample, you can do the following sequence of operations:

- Take edge $(1, 2)$, and remove vertices 1 and 2.
- Add vertex 6, connect it to vertex 5.
- Take edge $(4, 3)$, and remove vertices 4 and 3.
- Take edge $(5, 6)$, and remove vertices 5 and 6.

F. Fast XORting

Time limit: 1 second
Memory limit: 256 megabytes

You are given an integer n which is a power of two and a permutation $a_1, a_2, \ldots, a_n$ of $0, 1, \ldots, n-1$. In one operation you can do one of the following:

- Swap two adjacent elements. That is, choose any $1 \le i \le n-1$, and swap a_i, a_{i+1}
- Choose any integer $0 \le x \le n-1$, and replace a_i with a_i `XOR` x for every $1 \le i \le n$ (notice that the array remains a permutation)

What is the minimal number of operations needed to sort the permutation?

Here `XOR` denotes the bitwise XOR operation.

Input

The first line of the input contains a single integer n ($1 \le n \le 2^{18}$, n is a power of two) — the length of the permutation.

The second line contains n integers a_1, a_2, ..., a_n which form a permutation of 0, 1, ..., $n-1$.

Output

Output a single integer — the smallest number of operations needed to sort this permutation.

Examples

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

Note

In the first sample, we can sort the permutation with two operations as follows:

1. Swap a_1, a_2. The permutation becomes $[1, 0, 3, 2, 5, 4, 7, 6]$.
2. Choose $x = 1$, and `XOR` all elements with 1. It will become $[0, 1, 2, 3, 4, 5, 6, 7]$.

In the second sample, we can sort the permutation with two operations as follows:

1. Swap a_1, a_2. The permutation becomes $[0, 2, 1, 3, 4, 5, 6, 7]$.
2. Swap a_2, a_3. It will become $[0, 1, 2, 3, 4, 5, 6, 7]$.

G. Graph Race

Time limit: 2 seconds
Memory limit: 256 megabytes

You are given an unweighted undirected connected graph with n vertices and m edges. Each vertex u has two integers, a_u and b_u assigned to it. For each vertex v such that there exists an edge between 1 and v find:

$$\max_{u \neq v}\{a_u - b_u \cdot \text{dist}(u, v)\}$$

where $\text{dist}(u, v)$ denotes the distance between u and v.

Input

The first line of the standard input contains two integers n and m ($2 \leq n \leq 3 \cdot 10^5$, $1 \leq m \leq 3 \cdot 10^5$), respectively denoting the number of vertices of a graph and the number of its edges.

The following n lines contain two integers each a_u and b_u ($0 \leq a_u, b_u \leq 10^9$).

The following m lines contain two integers each u and v ($1 \leq u \neq v \leq n$), representing the edges of the graph. It is guaranteed that the graph doesn't contain multiple edges.

Output

In ascending order with respect to v such that there is an edge between 1 and v, print the value $\max_{u \neq v}\{a_u - b_u \cdot \text{dist}(u, v)\}$.

Example

standard input	standard output

5 4
0 0
1 1
1 1
5 1
100 40
4 1
1 2
1 3
4 5
	3
	3
	60

H. High Towers

Time limit: 2 seconds
Memory limit: 256 megabytes

You are given n towers in a row. The height of the i-th tower is h_i.

Towers can communicate with each other if one of them is higher than all the towers between them. More formally, towers i and j $(i < j)$ can communicate with each other if and only if $max(h_i, h_j) > max_{k \in [i+1, j-1]} h_k$. Note that adjacent towers always can communicate with each other.

For each tower, we know the value a_i — with how many other towers can i-th tower communicate. Find any possible sequence of heights h_i, $1 \le i \le n$.

It's guaranteed that in all provided tests there exists at least one possible sequence of heights. If there are multiple possible answers output any of them.

Input

The first line contains a single integer n $(2 \le n \le 5 \cdot 10^5)$ — the number of towers.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i \le n - 1)$ — the number of towers that can communicate with i-th tower.

Output

In a single line output n integers h_i $(1 \le h_i \le 10^9)$.

It's guaranteed that in all provided tests at least one possible sequence of h_i exists. If there are multiple possible answers output any of them.

Examples

standard input	standard output
6 3 3 4 2 5 1	7 5 7 1 10 4
4 3 3 3 3	3 2 1 4

Note

In the first sample, for $h = [7, 5, 7, 1, 10, 4]$:

- Tower 1 can communicate with towers $2, 3, 5$
- Tower 2 can communicate with towers $1, 3, 5$
- Tower 3 can communicate with towers $1, 2, 4, 5$
- Tower 4 can communicate with towers $3, 5$
- Tower 5 can communicate with towers $1, 2, 3, 4, 6$
- Tower 6 can communicate with tower 5

I. Impossible Numbers

Time limit: 5 seconds
Memory limit: 256 megabytes

You have received a calendar cube for your birthday! Fascinated by the fact that each day of the month could be constructed by using the two cubes in a specific orientation, you got an idea. You ordered n cubes online. Each cube has some digit written on each of its six faces. Digits may repeat within a cube.

Two number cubes forming the number 25.

Your curious mind begins to wonder: what are the k smallest numbers that **cannot** be obtained by using some of the n cubes in a specific orientation? Numbers must not contain leading zeros. Note that you can choose to not use some cube if you don't want to.

Input

The first line of the input contains two integers n and k ($1 \leq n \leq 100$, $1 \leq k \leq 10^5$).

Each of the following n lines contains exactly six numbers between 0 and 9 inclusively, representing the digits written on each of the six faces of the cubes.

Output

Output the smallest k positive numbers that cannot be obtained using the cubes, separated by space. The numbers must not contain leading zeros, and must be sorted **in increasing order**.

Examples

standard input	standard output

2 3
1 8 7 0 6 2
1 2 5 4 9 3
	33 34 35

1 10 1 5 2 2 6 4	3 7 8 9 10 11 12 13 14 15

4 10
1 5 7 1 2 4
0 1 5 8 9 4
3 5 2 2 7 8
6 1 7 0 2 2
	33 66 99 133 166 199 233 266 299 303

J. Jackpot

Time limit: 1 second
Memory limit: 256 megabytes

You are given an integer array a of $2n$ integers. In one operation, you can do the following:

- Choose any two adjacent elements in the array, let's say, a_i and a_{i+1}. Then, delete them, and add $|a_i - a_{i+1}|$ to your score.

  Note that the indices are recalculated after the operation.

You are going to perform this operation n times, deleting all the elements in the end. What is the largest score you can get?

Input

The first line contains a single integer t $(1 \le t \le 10^5)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n $(1 \le n \le 2 \cdot 10^5)$.

The second line of each test case contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ $(0 \le a_i \le 10^9)$ — elements of the array.

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

Output

For each test case, output a single integer — the largest possible score you can get after performing the operation n times.

Example

standard input	standard output

3
2
42 42 42 42
1
42 69
3
1 2 3 4 5 6
	0
	27
	9

Note

In the first test case, we can choose the first two elements, and delete them, getting score of 0, and then choose the remaining two elements, delete them, getting score of 0 again.

In the second test case, the only possible operation is to choose both elements and to get score of $|42 - 69| = 27$.

In the third test case, we can do the following sequence of operations:

1. Choose elements $3, 4$. Delete them, and get a score of 1. The array will become $[1, 2, 5, 6]$.
2. Choose elements $2, 5$. Delete them, and get a score of 3. The array will become $[1, 6]$.
3. Choose elements $1, 6$. Delete them, and get score of 5. The total score is $1 + 3 + 5 = 9$.

K. K Subsequences

Time limit: 1 second
Memory limit: 256 megabytes

For an array b, define $f(b)$ as the maximum sum on a subsegment of this array. For example, $f([-1,-1,-1]) = 0$, $f([-1,1,1,1,-1]) = 3$.

You are given an array a of length n, containing only 1s and -1s. Partition it into k subsequences $a_1, a_2, \ldots, a_k$ such that $max_{1 \le i \le k} f(a_i)$ is the minimum possible. If there are many solutions, output any.

Input

The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

The first line of each test case contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the length of the array and the number of subsequences.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($a_i = \pm 1$) — elements of the array.

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

Output

For each test case, output n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le k$). Here b_i means that element a_i belongs to the b_i-th subsequence.

Note that subsequences are allowed to be empty: it's allowed for some number $\le k$ to not appear in b.

Example

standard input	standard output

5
3 2
1 -1 1
4 2
-1 1 1 -1
7 3
1 1 1 1 1 1 1
10 3
1 1 1 1 -1 -1 1 1 1 1
12 4
1 1 1 1 -1 -1 -1 -1 1 1 1 1
	1 1 1
	1 1 2 2
	1 1 2 2 3 3 3
	1 2 1 2 1 2 1 2 3 3
	1 2 3 4 1 2 3 4 1 2 3 4

Note

In the first test case, we can put all elements into a single subsequence $[1,-1,1]$, with max subsegment sum 1 (the max subsegment sum for the remaining, empty subsequence is 0).

In the second test case, we can split elements into two subsequences $[-1,1], [1,-1]$, both with max subsegment sum 1.

In the third test case, we can split elements into three subsequences $[1,1], [1,1], [1,1,1]$, with max subsegment sums $2, 2, 3$ correspondingly.

In the fourth test case, we can split elements into three subsequences $[1,1,-1,1], [1,1,-1,1], [1,1]$, all with max subsegment sum 2.

In the fifth test case, we can split elements into four subsequences $[1,-1,1], [1,-1,1], [1,-1,1], [1,-1,1]$, all with max subsegment sum 1.

L. LIS on Grid

Time limit: 1 second
Memory limit: 256 megabytes

You are given a grid with n rows and m columns. n rows are numbered from 1 to n from top to bottom, and m columns are numbered from 1 to m from left to right. Let's denote the cell at the intersection of the i-th row and the j-th column by (i, j).

You have to color some of these cells in black. More precisely, in i-th column you should color exactly a_i cells in black.

After you color cells of your choice, your penalty will be calculated as the length of the longest increasing subsequence of black cells. More precisely, it will be equal to the largest k, for which there exists a sequence of cells $(r_1, c_1), (r_2, c_2), \ldots, (r_k, c_k)$, such that:

- $1 \le r_1 < r_2 < \ldots < r_k \le n$
- $1 \le c_1 < c_2 < \ldots < c_k \le m$
- All cells (r_i, c_i) are black

Find the smallest possible penalty you can get.

Input

The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

The first line of each test case contains two integers n, m ($1 \le n, m \le 2 \cdot 10^5, m \cdot n \le 2 \cdot 10^5$) — the numbers of rows and columns correspondingly.

The second line of each test case contains m integers $a_1, a_2, \ldots, a_m$ ($1 \le a_i \le n$), indicating that you have to color exactly a_i cells in column i.

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

Output

For each test case, in the first line output a single integer — the smallest penalty you can achieve.

In the next n lines, output n strings. i-th string should have length m and consist only of characters `.` and `#`. If j-th character of i-th string is `#`, cell (i, j) is colored black, otherwise it's colored white.

Example

standard input	standard output

`4`
`2 4`
`1 1 1 1`
`3 3`
`3 3 3`
`4 4`
`4 3 2 1`
`4 5`
`2 3 4 3 2`
	`1`
	`....`
	`####`
	`3`
	`###`
	`###`
	`###`
	`2`
	`###.`
	`#...`
	`####`
	`##..`
	`2`
	`..###`
	`.####`
	`####.`
	`###..`

M. Max Minus Min

Time limit: 1 second
Memory limit: 256 megabytes

You are given array a of n integers. You can perform the following operation at most once:

- Choose any integers $1 \le l \le r \le n$, and any integer x. Then, for each $l \le i \le r$, replace a_i with $a_i + x$.

Find the smallest possible value of $\max(a_1, a_2, \ldots, a_n) - \min(a_1, a_2, \ldots, a_n)$ you can get after performing this operation at most once.

Input

The first line contains a single integer t $(1 \le t \le 10^5)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n $(1 \le n \le 2 \cdot 10^5)$ — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i \le 10^9)$ — elements of the array.

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

Output

For each test case, in the first line output a single integer — the smallest possible value of $\max(a_1, a_2, \ldots, a_n) - \min(a_1, a_2, \ldots, a_n)$ you can get after performing this operation at most once.

Example

standard input	standard output

4
3
42 42 42
4
1 2 2 1
5
1 100 1 100 1
6
1 2 3 4 5 6
	0
	0
	99
	2

Note

In the first test case, you don't need to make any operations, since $max - min = 0$ already.

In the second test case, you can choose $x = -1$, and segment $a[2:3]$. The array will become $[1, 1, 1, 1]$, with $max - min = 0$.

In the third test case, $max - min$ initially is 99. Unfortunately, it's not possible to decrease this value with a single operation.

In the fourth test case, $max - min$ initially is 5, but we can choose $x = 3$ and segment $a[1:3]$. The array will become $[4, 5, 6, 4, 5, 6]$, with $max - min = 2$.