A. Another One Bites The Dust

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Let's call a string *good* if and only if it consists of only two types of letters — 'a' and 'b' and every two consecutive letters are distinct. For example "baba" and "aba" are good strings and "abb" is a bad string.

You have a strings "a", b strings "b" and c strings "ab". You want to choose some subset of these strings and concatenate them in any arbitrarily order.

What is the length of the longest good string you can obtain this way?

Input

The first line contains three positive integers a, b, c ($1 \leq a, b, c \leq 10^9$) — the number of strings "a", "b" and "ab" respectively.

Output

Print a single number — the maximum possible length of the good string you can obtain.

Examples

input
```
1 1 1
```
output
```
4
```

input
```
2 1 2
```
output
```
7
```

input
```
3 5 2
```
output
```
11
```

input
```
2 2 1
```
output
```
6
```

input
```
1000000000 1000000000 1000000000
```
output
```
4000000000
```

Note

In the first example the optimal string is "baba".

In the second example the optimal string is "abababa".

In the third example the optimal string is "bababababab".

In the fourth example the optimal string is "ababab".

B. Born This Way

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input

output: standard output

Arkady bought an air ticket from a city A to a city C. Unfortunately, there are no direct flights, but there are a lot of flights from A to a city B, and from B to C.

There are n flights from A to B, they depart at time moments a_1, a_2, a_3, ..., a_n and arrive at B t_a moments later.

There are m flights from B to C, they depart at time moments b_1, b_2, b_3, ..., b_m and arrive at C t_b moments later.

The connection time is negligible, so one can use the i-th flight from A to B and the j-th flight from B to C if and only if $b_j \geq a_i + t_a$.

You can cancel at most k flights. If you cancel a flight, Arkady can not use it.

Arkady wants to be in C as early as possible, while you want him to be in C as late as possible. Find the earliest time Arkady can arrive at C, if you optimally cancel k flights. If you can cancel k or less flights in such a way that it is not possible to reach C at all, print -1.

Input

The first line contains five integers n, m, t_a, t_b and k ($1 \leq n, m \leq 2 \cdot 10^5$, $1 \leq k \leq n + m$, $1 \leq t_a, t_b \leq 10^9$) — the number of flights from A to B, the number of flights from B to C, the flight time from A to B, the flight time from B to C and the number of flights you can cancel, respectively.

The second line contains n distinct integers in increasing order a_1, a_2, a_3, ..., a_n ($1 \leq a_1 < a_2 < \ldots < a_n \leq 10^9$) — the times the flights from A to B depart.

The third line contains m distinct integers in increasing order b_1, b_2, b_3, ..., b_m ($1 \leq b_1 < b_2 < \ldots < b_m \leq 10^9$) — the times the flights from B to C depart.

Output

If you can cancel k or less flights in such a way that it is not possible to reach C at all, print -1.

Otherwise print the earliest time Arkady can arrive at C if you cancel k flights in such a way that maximizes this time.

Examples

input

```
4 5 1 1 2
1 3 5 7
1 2 3 9 10
```

output

```
11
```

input

```
2 2 4 4 2
1 10
10 20
```

output

```
-1
```

input

```
4 3 2 3 1
1 999999998 999999999 1000000000
3 4 1000000000
```

output

```
1000000003
```

Note

Consider the first example. The flights from A to B depart at time moments 1, 3, 5, and 7 and arrive at B at time moments 2, 4, 6, 8, respectively. The flights from B to C depart at time moments 1, 2, 3, 9, and 10 and arrive at C at time moments 2, 3, 4, 10, 11, respectively. You can cancel at most two flights. The optimal solution is to cancel the first flight from A to B and the fourth flight from B to C. This way Arkady has to take the second flight from A to B, arrive at B at time moment 4, and take the last flight from B to C arriving at C at time moment 11.

In the second example you can simply cancel all flights from A to B and you're done.

In the third example you can cancel only one flight, and the optimal solution is to cancel the first flight from A to B. Note that there is still just enough time to catch the last flight from B to C.

C. Crazy Diamond

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given a permutation p of integers from 1 to n, where n is an even number.

Your goal is to sort the permutation. To do so, you can perform zero or more operations of the following type:

- take two indices i and j such that $2 \cdot |i - j| \geq n$ and swap p_i and p_j.

There is **no need to minimize** the number of operations, however you should use no more than $5 \cdot n$ operations. One can show that it is always possible to do that.

Input

The first line contains a single integer n ($2 \leq n \leq 3 \cdot 10^5$, n is even) — the length of the permutation.

The second line contains n distinct integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$) — the given permutation.

Output

On the first line print m ($0 \leq m \leq 5 \cdot n$) — the number of swaps to perform.

Each of the following m lines should contain integers a_i, b_i ($1 \leq a_i, b_i \leq n$, $|a_i - b_i| \geq \frac{n}{2}$) — the indices that should be swapped in the corresponding swap.

Note that there is no need to minimize the number of operations. We can show that an answer always exists.

Examples

input
```
2
2 1
```
output
```
1
1 2
```

input
```
4
3 4 1 2
```
output
```
4
1 4
1 4
1 3
2 4
```

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

Note

In the first example, when one swap elements on positions 1 and 2, the array becomes sorted.

In the second example, pay attention that there is no need to minimize number of swaps.

In the third example, after swapping elements on positions 1 and 5 the array becomes: $[4, 5, 3, 1, 2, 6]$. After swapping elements on positions 2 and 5 the array becomes $[4, 2, 3, 1, 5, 6]$ and finally after swapping elements on positions 1 and 4 the array becomes sorted: $[1, 2, 3, 4, 5, 6]$.

D. Dirty Deeds Done Dirt Cheap

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n pairs of integers $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$. All of the integers in the pairs are distinct and are in the range from 1 to $2 \cdot n$ inclusive.

Let's call a sequence of integers $x_1, x_2, \ldots, x_{2k}$ *good* if either

- $x_1 < x_2 > x_3 < \ldots < x_{2k-2} > x_{2k-1} < x_{2k}$, or
- $x_1 > x_2 < x_3 > \ldots > x_{2k-2} < x_{2k-1} > x_{2k}$.

You need to choose a subset of distinct indices $i_1, i_2, \ldots, i_t$ and **their order** in a way that if you write down all numbers from the pairs in a single sequence (the sequence would be $a_{i_1}, b_{i_1}, a_{i_2}, b_{i_2}, \ldots, a_{i_t}, b_{i_t}$), this sequence is good.

What is the largest subset of indices you can choose? You also need to construct the corresponding index sequence $i_1, i_2, \ldots, i_t$.

Input

The first line contains single integer n ($2 \leq n \leq 3 \cdot 10^5$) — the number of pairs.

Each of the next n lines contain two numbers — a_i and b_i ($1 \leq a_i, b_i \leq 2 \cdot n$) — the elements of the pairs.

It is guaranteed that all integers in the pairs are distinct, that is, every integer from 1 to $2 \cdot n$ is mentioned exactly once.

Output

In the first line print a single integer t — the number of pairs in the answer.

Then print t distinct integers $i_1, i_2, \ldots, i_t$ — the indexes of pairs in the corresponding order.

Examples

input

```
5
1 7
6 4
2 10
9 8
3 5
```

output

```
3
1 5 3
```

input

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

output

```
3
3 2 1
```

Note

The final sequence in the first example is $1 < 7 > 3 < 5 > 2 < 10$.

The final sequence in the second example is $6 > 1 < 3 > 2 < 5 > 4$.

E. Earth Wind and Fire

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n stones arranged on an axis. Initially the i-th stone is located at the coordinate s_i. There may be more than one stone in a single place.

You can perform zero or more operations of the following type:

- take two stones with indices i and j so that $s_i \leq s_j$, choose an integer d ($0 \leq 2 \cdot d \leq s_j - s_i$), and replace the coordinate s_i with $(s_i + d)$ and replace coordinate s_j with $(s_j - d)$. In other words, draw stones closer to each other.

You want to move the stones so that they are located at positions $t_1, t_2, \ldots, t_n$. The order of the stones is not important — you just want for the multiset of the stones resulting positions to be the same as the multiset of $t_1, t_2, \ldots, t_n$.

Detect whether it is possible to move the stones this way, and if yes, construct a way to do so. You don't need to minimize the number of moves.

Input

The first line contains a single integer n ($1 \leq n \leq 3 \cdot 10^5$) – the number of stones.

The second line contains integers $s_1, s_2, \ldots, s_n$ ($1 \leq s_i \leq 10^9$) — the initial positions of the stones.

The second line contains integers $t_1, t_2, \ldots, t_n$ ($1 \leq t_i \leq 10^9$) — the target positions of the stones.

Output

If it is impossible to move the stones this way, print "NO".

Otherwise, on the first line print "YES", on the second line print the number of operations m ($0 \leq m \leq 5 \cdot n$) required. You don't have to minimize the number of operations.

Then print m lines, each containing integers i, j, d ($1 \leq i, j \leq n$, $s_i \leq s_j$, $0 \leq 2 \cdot d \leq s_j - s_i$), defining the operations.

One can show that if an answer exists, there is an answer requiring no more than $5 \cdot n$ operations.

Examples

input

```
5
2 2 7 4 9
5 4 5 5 5
```

output

```
YES
4
4 3 1
2 3 1
2 5 2
1 5 2
```

input

```
3
1 5 10
3 5 7
```

output

```
NO
```

Note

Consider the first example.

- After the first move the locations of stones is $[2, 2, 6, 5, 9]$.
- After the second move the locations of stones is $[2, 3, 5, 5, 9]$.
- After the third move the locations of stones is $[2, 5, 5, 5, 7]$.
- After the last move the locations of stones is $[4, 5, 5, 5, 5]$.

F. Foo Fighters

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n objects. Each object has two integer properties: val_i — its price — and $mask_i$. It is guaranteed that the sum of all prices is initially non-zero.

You want to select a positive integer s. All objects will be modified after that. The i-th object will be modified using the following procedure:

- Consider $mask_i$ and s in binary notation,
- Compute the bitwise AND of s and $mask_i$ ($s \,\&\, mask_i$),
- If ($s \,\&\, mask_i$) contains an odd number of ones, replace the val_i with $-val_i$. Otherwise do nothing with the i-th object.

You need to find such an integer s that when the modification above is done the sum of all prices changes sign (if it was negative, it should become positive, and vice-versa; it is not allowed for it to become zero). The absolute value of the sum can be arbitrary.

Input

The first line contains a single integer n ($1 \le n \le 3 \cdot 10^5$) — the number of objects.

The i-th of next n lines contains integers val_i and $mask_i$ ($-10^9 \le val_i \le 10^9$, $1 \le mask_i \le 2^{62} - 1$) — the price of the object and its mask.

It is guaranteed that the sum of val_i is initially non-zero.

Output

Print an integer s ($1 \le s \le 2^{62} - 1$), such that if we modify the objects as described above, the sign of the sum of val_i changes its sign.

If there are multiple such s, print any of them. One can show that there is always at least one valid s.

Examples

input

```
5
17 206
-6 117
-2 151
9 93
6 117
```

output

```
64
```

input

```
1
1 1
```

output

```
1
```

Note

In the first test sample all objects will change their prices except for the object with mask 151.

So their total sum will change its sign: initially 24, after modifications — -28.

In the second test sample the only object will change its price. So the total sum will change its sign.

G. Gold Experience

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Consider an undirected graph G with n vertices. There is a value a_i in each vertex.

Two vertices i and j are connected with an edge if and only if $gcd(a_i, a_j) > 1$, where $gcd(x, y)$ denotes the greatest common divisor (GCD) of integers x and y.

Consider a set of vertices. Let's call a vertex in this set *fair* if it is connected with an edge with all other vertices in this set.

You need to find a set of k vertices (where k is a given integer, $2 \cdot k \le n$) where all vertices are fair or all vertices are not fair. One can show that such a set always exists.

Input

The first line contains integers n and k ($6 \le 2 \cdot k \le n \le 10^5$) — the number of vertices and parameter k.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($2 \le a_i \le 10^7$) — the values in the vertices.

Output

Print exactly k distinct integers — the indices of the vertices in the chosen set in any order.

Examples

input

```
6 3
6 15 10 8 14 12
```

output

```
2 4 5
```

input

```
8 4
11 15 10 6 21 15 10 6
```

output

```
5 7 1 2
```

input

```
10 5
3003 17017 3230 49742 546 41990 17765 570 21945 36465
```

output

```
1 2 4 5 6
```

Note

In the first test case, set $\{2, 4, 5\}$ is an example of set where no vertices are *fair*. The vertex 2 does not share an edge with vertex 4 since $gcd(15, 8) = 1$. The vertex 4 does not share an edge with vertex 2. The vertex 5 does not share an edge with vertex 2.

In the second test case, set $\{8, 5, 6, 4\}$ is an example of a set where all vertices are *fair*.

H. Holy Diver

time limit per test: 3 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You are given an array which is initially empty. You need to perform n operations of the given format:

- "$a\ l\ r\ k$": append a to the end of the array. After that count the number of integer pairs x, y such that $l \le x \le y \le r$ and

$\text{mex}(a_x, a_{x+1}, \ldots, a_y) = k.$

The elements of the array are numerated from 1 in the order they are added to the array.

To make this problem more tricky we don't say your real parameters of the queries. Instead your are given a', l', r', k'. To get a, l, r, k on the i-th operation you need to perform the following:

- $a := (a' + lans) \bmod (n+1)$,
- $l := (l' + lans) \bmod i + 1$,
- $r := (r' + lans) \bmod i + 1$,
- if $l > r$ swap l and r,
- $k := (k' + lans) \bmod (n+1)$,

where $lans$ is the answer to the previous operation, initially $lans$ is equal to zero. i is the id of the operation, operations are numbered from 1.
The $\text{mex}(S)$, where S is a multiset of non-negative integers, is the smallest non-negative integer which does not appear in the set. For example, $\text{mex}(\{2, 2, 3\}) = 0$ and $\text{mex}(\{0, 1, 4, 1, 6\}) = 2$.

Input

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

The next n lines contain the description of queries.

Each of them n lines contains four non-negative integers a', l', r', k' ($0, \le a', l', r', k' \le 10^9$), describing one operation.

Output

For each query print a single integer — the answer to this query.

Examples

input

```
5
0 0 0 1
0 1 0 5
5 2 1 0
5 2 1 0
2 4 3 3
```

output

```
1
1
2
6
3
```

input

```
5
2 0 0 2
2 0 1 1
0 0 2 0
3 2 2 0
0 2 3 0
```

output

```
0
0
3
0
0
```

Note

For the first example the decoded values of a, l, r, k are the following:

$a_1 = 0, l_1 = 1, r_1 = 1, k_1 = 1$

$a_2 = 1, l_2 = 1, r_2 = 2, k_2 = 0$

$a_3 = 0, l_3 = 1, r_3 = 3, k_3 = 1$

$a_4 = 1, l_4 = 1, r_4 = 4, k_4 = 2$

$a_5 = 2, l_5 = 1, r_5 = 5, k_5 = 3$

For the second example the decoded values of a, l, r, k are the following:

$a_1 = 2, l_1 = 1, r_1 = 1, k_1 = 2$

$a_2 = 2, l_2 = 1, r_2 = 2, k_2 = 1$

$a_3 = 0, l_3 = 1, r_3 = 3, k_3 = 0$

$a_4 = 0, l_4 = 2, r_4 = 2, k_4 = 3$

$a_5 = 0, l_5 = 3, r_5 = 4, k_5 = 0$