## A. Countdown

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

You are given a digital clock with $n$ digits. Each digit shows an integer from $0$ to $9$, so the whole clock shows an integer from $0$ to $10^n - 1$. The clock will show leading zeroes if the number is smaller than $10^{n-1}$.

You want the clock to show $0$ with as few operations as possible. In an operation, you can do one of the following:

- decrease the number on the clock by $1$, or
- swap two digits (you can choose which digits to swap, and they don't have to be adjacent).

Your task is to determine the minimum number of operations needed to make the clock show $0$.

### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 10^3$).

The first line of each test case contains a single integer $n$ ($1 \le n \le 100$) — number of digits on the clock.

The second line of each test case contains a string of $n$ digits $s_1, s_2, \ldots, s_n$ ($0 \le s_1, s_2, \ldots, s_n \le 9$) — the number on the clock.

Note: If the number is smaller than $10^{n-1}$ the clock will show leading zeroes.

### Output

For each test case, print one integer: the minimum number of operations needed to make the clock show $0$.

### Example

**input**

```
7
3
007
4
1000
5
00000
3
103
4
2020
9
123456789
30
001678294039710047203946100020
```

**output**

```
7
2
0
5
6
53
115
```

### Note

In the first example, it's optimal to just decrease the number $7$ times.

In the second example, we can first swap the first and last position and then decrease the number by $1$.

In the third example, the clock already shows $0$, so we don't have to perform any operations.

## B. Swaps

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

You are given two arrays $a$ and $b$ of length $n$. Array $a$ contains each **odd** integer from $1$ to $2n$ in an arbitrary order, and array $b$ contains each **even** integer from $1$ to $2n$ in an arbitrary order.

You can perform the following operation on those arrays:

- choose one of the two arrays
- pick an index $i$ from $1$ to $n-1$
- swap the $i$-th and the $(i+1)$-th elements of the chosen array

Compute the minimum number of operations needed to make array $a$ lexicographically smaller than array $b$.
For two different arrays $x$ and $y$ of the same length $n$, we say that $x$ is lexicographically smaller than $y$ if in the first position where $x$ and $y$ differ, the array $x$ has a smaller element than the corresponding element in $y$.

## Input

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

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

The second line of each test case contains $n$ integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 2n$, all $a_i$ are **odd** and pairwise distinct) — array $a$.

The third line of each test case contains $n$ integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 2n$, all $b_i$ are **even** and pairwise distinct) — array $b$.

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

## Output

For each test case, print one integer: the minimum number of operations needed to make array $a$ lexicographically smaller than array $b$.

We can show that an answer always exists.

### Example

**input**

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

**output**

```
0
2
3
```

## Note

In the first example, the array $a$ is already lexicographically smaller than array $b$, so no operations are required.

In the second example, we can swap $5$ and $3$ and then swap $2$ and $4$, which results in $[3, 5, 1]$ and $[4, 2, 6]$. Another correct way is to swap $3$ and $1$ and then swap $5$ and $1$, which results in $[1, 5, 3]$ and $[2, 4, 6]$. Yet another correct way is to swap $4$ and $6$ and then swap $2$ and $6$, which results in $[5, 3, 1]$ and $[6, 2, 4]$.

# C. Book

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a book with $n$ chapters.

Each chapter has a specified list of other chapters that need to be understood in order to understand this chapter. To understand a chapter, you must read it after you understand every chapter on its required list.

Currently you don't understand any of the chapters. You are going to read the book from the beginning till the end repeatedly until you understand the whole book. Note that if you read a chapter at a moment when you don't understand some of the required chapters, you don't understand this chapter.

Determine how many times you will read the book to understand every chapter, or determine that you will never understand every chapter no matter how many times you read the book.

## Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 2 \cdot 10^4$).

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

Then $n$ lines follow. The $i$-th line begins with an integer $k_i$ ($0 \le k_i \le n-1$) — number of chapters required to understand the $i$-th chapter. Then $k_i$ integers $a_{i,1}, a_{i,2}, \ldots, a_{i,k_i}$ ($1 \le a_{i,j} \le n, a_{i,j} \ne i, a_{i,j} \ne a_{i,l}$ for $j \ne l$) follow — the chapters required to understand the $i$-th chapter.

It is guaranteed that the sum of $n$ and sum of $k_i$ over all testcases do not exceed $2 \cdot 10^5$.

### Output

For each test case, if the entire book can be understood, print how many times you will read it, otherwise print $-1$.

**Example**

**input**

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

**output**

```
2
-1
1
2
5
```

### Note

In the first example, we will understand chapters $\{2, 4\}$ in the first reading and chapters $\{1, 3\}$ in the second reading of the book.

In the second example, every chapter requires the understanding of some other chapter, so it is impossible to understand the book.

In the third example, every chapter requires only chapters that appear earlier in the book, so we can understand everything in one go.

In the fourth example, we will understand chapters $\{2, 3, 4\}$ in the first reading and chapter $1$ in the second reading of the book.

In the fifth example, we will understand one chapter in every reading from $5$ to $1$.

## D. Xor of 3

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

You are given a sequence $a$ of length $n$ consisting of $0$s and $1$s.

You can perform the following operation on this sequence:

- Pick an index $i$ from $1$ to $n-2$ (inclusive).
- Change all of $a_i$, $a_{i+1}$, $a_{i+2}$ to $a_i \oplus a_{i+1} \oplus a_{i+2}$ simultaneously, where $\oplus$ denotes the bitwise XOR operation

Find a sequence of **at most** $n$ operations that changes all elements of $a$ to $0$s or report that it's impossible.
We can prove that if there exists a sequence of operations of any length that changes all elements of $a$ to $0$s, then there is also such a sequence of length not greater than $n$.

### Input

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

The first line of each test case contains a single integer $n$ ($3 \le n \le 2 \cdot 10^5$) — the length of $a$.

The second line of each test case contains $n$ integers $a_1, a_2, \ldots, a_n$ ($a_i = 0$ or $a_i = 1$) — elements of $a$.

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

### Output

For each test case, do the following:

- if there is no way of making all the elements of $a$ equal to $0$ after performing the above operation some number of times, print "NO".
- otherwise, in the first line print "YES", in the second line print $k$ ($0 \le k \le n$) — the number of operations that you want to perform on $a$, and in the third line print a sequence $b_1, b_2, \dots, b_k$ ($1 \le b_i \le n-2$) — the indices on which the operation should be applied.

If there are multiple solutions, you may print any.

#### Example

**input**

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

**output**

```
YES
0
YES
2
3 1
NO
```

### Note

In the first example, the sequence contains only $0$s so we don't need to change anything.

In the second example, we can transform $[1, 1, 1, 1, 0]$ to $[1, 1, 0, 0, 0]$ and then to $[0, 0, 0, 0, 0]$ by performing the operation on the third element of $a$ and then on the first element of $a$.

In the third example, no matter whether we first perform the operation on the first or on the second element of $a$ we will get $[1, 1, 1, 1]$, which cannot be transformed to $[0, 0, 0, 0]$.

## E. Paint

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

You are given a $1$ by $n$ pixel image. The $i$-th pixel of the image has color $a_i$. For each color, the number of pixels of that color is **at most** $20$.

You can perform the following operation, which works like the bucket tool in paint programs, on this image:

- pick a color — an integer from $1$ to $n$;
- choose a pixel in the image;
- for all pixels connected to the selected pixel, change their colors to the selected color (two pixels of the same color are considered connected if all the pixels between them have the same color as those two pixels).

Compute the minimum number of operations needed to make all the pixels in the image have the same color.

### Input

Each test contains multiple test cases. The first line contains the number of test cases $t$ ($1 \le t \le 10^3$).

The first line of each test case contains a single integer $n$ ($1 \le n \le 3 \cdot 10^3$) — the number of pixels in the image.

The second line of each test case contains $n$ integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — the colors of the pixels in the image.

Note: for each color, the number of pixels of that color is **at most** $20$.

It is guaranteed that the sum of $n$ over all test cases does not exceed $3 \cdot 10^3$.

### Output

For each test case, print one integer: the minimum number of operations needed to make all the pixels in the image have the same color.

#### Example

**input**

```
3
5
```

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

**output**

```
2
1
3
```

### Note

In the first example, the optimal solution is to apply the operation on the third pixel changing its color to $2$ and then to apply the operation on any pixel that has color $2$ changing its color and the color of all pixels connected to it to $1$. The sequence of operations is then: $[1, 2, 3, 2, 1] \rightarrow [1, 2, 2, 2, 1] \rightarrow [1, 1, 1, 1, 1]$.

In the second example, we can either change the $1$s to $2$s in one operation or change the $2$s to $1$s also in one operation.

In the third example, one possible way to make all the pixels have the same color is to apply the operation on the first, third and the fourth pixel each time changing its color to $2$.

## F. Bridge Club

time limit per test: 2.5 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

There are currently $n$ hot topics numbered from $0$ to $n-1$ at your local bridge club and $2^n$ players numbered from $0$ to $2^n-1$. Each player holds a different set of views on those $n$ topics, more specifically, the $i$-th player holds a positive view on the $j$-th topic if $i \ \& \ 2^j > 0$, and a negative view otherwise. Here $\&$ denotes the bitwise AND operation.

You are going to organize a bridge tournament capable of accommodating at most $k$ pairs of players (bridge is played in teams of two people). You can select teams arbitrarily while each player is in at most one team, but there is one catch: two players cannot be in the same pair if they disagree on $2$ or more of those $n$ topics, as they would argue too much during the play.

You know that the $i$-th player will pay you $a_i$ dollars if they play in this tournament. Compute the maximum amount of money that you can earn if you pair the players in your club optimally.

### Input

The first line contains two integers $n$, $k$ ($1 \leq n \leq 20$, $1 \leq k \leq 200$) — the number of hot topics and the number of pairs of players that your tournament can accommodate.

The second line contains $2^n$ integers $a_0, a_1, \ldots, a_{2^n-1}$ ($0 \leq a_i \leq 10^6$) — the amounts of money that the players will pay to play in the tournament.

### Output

Print one integer: the maximum amount of money that you can earn if you pair the players in your club optimally under the above conditions.

### Examples

**input**

```
3 1
8 3 5 7 1 10 3 2
```

**output**

```
13
```

**input**

```
2 3
7 4 5 7
```

**output**

```
23
```

**input**

```
3 2
1 9 1 5 7 8 1 1
```

**output**

```
29
```

### Note

In the first example, the best we can do is to pair together the $0$-th player and the $2$-nd player resulting in earnings of $8+5=13$ dollars. Although pairing the $0$-th player with the $5$-th player would give us $8+10=18$ dollars, we cannot do this because those two players disagree on $2$ of the $3$ hot topics.

In the second example, we can pair the $0$-th player with the $1$-st player and pair the $2$-nd player with the $3$-rd player resulting in earnings of $7 + 4 + 5 + 7 = 23$ dollars.