Codeforces Round 876 (Div. 2)

A. The Good Array

1 second, 256 megabytes

You are given two integers n and k.

An array $a_1, a_2, \ldots, a_n$ of length n, consisting of zeroes and ones is *good* if for **all** integers i from 1 to n **both** of the following conditions are satisfied:

- at least $\lceil \frac{i}{k} \rceil$ of the first i elements of a are equal to 1,
- at least $\lceil \frac{i}{k} \rceil$ of the last i elements of a are equal to 1.

Here, $\lceil \frac{i}{k} \rceil$ denotes the result of division of i by k, rounded up. For example, $\lceil \frac{6}{3} \rceil = 2$, $\lceil \frac{11}{5} \rceil = \lceil 2.2 \rceil = 3$ and $\lceil \frac{7}{4} \rceil = \lceil 1.75 \rceil = 2$.

Find the minimum possible number of ones in a good array.

Input

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

The only line of each test case contains two integers n, k ($2 \le n \le 100$, $1 \le k \le n$) — the length of array and parameter k from the statement.

Output

For each test case output one integer — the minimum possible number of ones in a good array.

It can be shown that under the given constraints at least one good array always exists.

input
7 3 2 5 2 9 3 7 1 10 4 9 5 8 8

output
2 3 4 7 4 3 2

In the first test case, $n = 3$ and $k = 2$:

- Array $[1, 0, 1]$ is good and the number of ones in it is 2.
- Arrays $[0, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$ are not good since for $i = 1$ the first condition from the statement is not satisfied.
- Array $[1, 0, 0]$ is not good since for $i = 1$ the second condition from the statement is not satisfied.
- All other arrays of length 3 contain at least 2 ones.

Thus, the answer is 2.

In the second test case, $n = 5$ and $k = 2$:

- Array $[1, 1, 0, 0, 1]$ is not good since for $i = 3$ the second condition is not satisfied.
- Array $[1, 0, 1, 0, 1]$ is good and the number of ones in it is 3.
- It can be shown that there is no good array with less than 3 ones, so the answer is 3.

In the third test case, $n = 9$ and $k = 3$:

- Array $[1, 0, 1, 0, 0, 0, 1, 0, 1]$ is good and the number of ones in it is 4.
- It can be shown that there is no good array with less than 4 ones, so the answer is 4.

In the fourth test case, $n = 7$ and $k = 1$. The only good array is $[1, 1, 1, 1, 1, 1, 1]$, so the answer is 7.

B. Lamps

1 second, 256 megabytes

You have n lamps, numbered by integers from 1 to n. Each lamp i has two integer parameters a_i and b_i.

At each moment each lamp is **in one of three states**: it may be turned on, turned off, or broken.

Initially all lamps are turned off. In one operation you can select one lamp that is turned off and turn it on (you can't turn on broken lamps). You **receive** b_i points for turning lamp i on. The following happens after each performed operation:

- Let's denote the number of lamps that are turned on as x (broken lamps **do not count**). All lamps i such that $a_i \le x$ simultaneously break, whether they were turned on or off.

Please note that broken lamps never count as turned on and that after a turned on lamp breaks, you still keep points received for turning it on.

You can perform an arbitrary number of operations.

Find the maximum number of points you can get.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

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

Each of the next n lines contains two integers a_i and b_i ($1 \le a_i \le n, 1 \le b_i \le 10^9$) — parameters of the i-th lamp.

It is guaranteed that sum of n over all test cases doesn't exceed $2 \cdot 10^5$.

Output

For each test case, output one integer — the maximum number of points you can get.

input
4 4 2 2 1 6 1 10 1 13 5 3 4 3 1 2 5 3 2 3 3 6 1 2 3 4 1 4 3 4 3 5 2 3 1 1 1

output

```
15
14
20
1
```

In first test case $n = 4$. One of ways to get the maximum number of points is as follows:

- You turn lamp 4 on and receive $b_4 = 13$ points.
- The number of lamps that are turned on is 1, so all lamps with $a_i \le 1$ (namely lamps 2, 3 and 4) break. Lamp 4 is no longer turned on, so the number of lamps that are turned becomes 0.
- The only lamp you can turn on is lamp 1, as all other lamps are broken. You receive $b_1 = 2$ points for turning it on.
- The number of lamps that are turned on is 1. As $a_1 = 2$, lamp 1 doesn't break.

Your receive $13 + 2 = 15$ points in total. It can be shown that this is the maximum number of points you can get, so the answer for the first test case is 15.

In the second test case, one of the ways to get the maximum number of points is as follows:

- On the first operation you turn on lamp 4 and receive 2 points. No lamps break after the first operation.
- On the second operation you turn on lamp 3 and receive 5 points. After the second operation, there are 2 lamps turned on. As $a_3 \le 2$, lamp 3 breaks.
- On the third operation, you turn on lamp 1 and receive 4 points.
- On the fourth operation, you turn on lamp 5 and receive 3 points. After that there are 3 lamps turned on: lamps 1, 4 and 5. Lamps 1, 2, 4 and 5 simultaneously break, because for all of them $a_i \le 3$.

You receive $2 + 5 + 4 + 3 = 14$ points in total. It can be shown that this is the maximum number of points you can get.

In the third test case, one of the ways to get the maximum number of points is as follows:

- Turn the lamp 3 on and receive 4 points. Lamps 1 and 3 break.
- Turn the lamp 2 on and receive 4 points.
- Turn the lamp 6 on and receive 3 points. Lamp 6 breaks.
- Turn the lamp 4 on and receive 4 points.
- Turn the lamp 5 on and receive 5 points. Lamps 2, 4 and 5 break.

You receive $4 + 4 + 3 + 4 + 5 = 20$ points in total. It can be shown that this is the maximum number of points you can get.

C. Insert Zero and Invert Prefix

2 seconds, 256 megabytes

You have a sequence $a_1, a_2, \ldots, a_n$ of length n, each element of which is either 0 or 1, and a sequence b, which is initially empty.

You are going to perform n operations. On each of them you will increase the length of b by 1.

- On the i-th operation you choose an integer p between 0 and $i - 1$. You insert 0 in the sequence b on position $p + 1$ (after the first p elements), and then you invert the first p elements of b.
- More formally: let's denote the sequence b before the i-th ($1 \le i \le n$) operation as $b_1, b_2, \ldots, b_{i-1}$. On the i-th operation you choose an integer p between 0 and $i - 1$ and replace b with $\overline{b_1}, \overline{b_2}, \ldots, \overline{b_p}, 0, b_{p+1}, b_{p+2}, \ldots, b_{i-1}$. Here, $\overline{x}$ denotes the binary inversion. Hence, $\overline{0} = 1$ and $\overline{1} = 0$.

You can find examples of operations in the Notes section.

Determine if there exists a sequence of operations that makes b equal to a. If such sequence of operations exists, find it.

Input

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

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

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

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

Output

For each test case:

- output "NO", if it is impossible to make b equal to a using the given operations;
- otherwise, output "YES" in the first line and n integers $p_1, p_2, \ldots, p_n$ ($0 \le p_i \le i - 1$) in the second line — the description of sequence of operations that makes b equal to a. Here, p_i should be the integer you choose on the i-th operation. If there are multiple solutions, you can output **any** of them.

input

```
4
5
1 1 0 0 0
1
1
3
0 1 1
6
1 0 0 1 1 0
```

output

```
YES
0 0 2 1 3
NO
NO
YES
0 1 0 2 4 2
```

In the first test case,

1. Before the first operation, $b = [\,]$. You choose $p = 0$ and replace b with $[\,\underline{0}\,]$
2. On the second operation you choose $p = 0$ and replace b with $[\,\underline{0}, 0\,]$.
3. On the third operation you choose $p = 2$ and replace b with $[\,1, 1, \underline{0}\,]$.
4. On the fourth operation you choose $p = 1$ and replace b with $[\,0, \underline{0}, 1, 0\,]$.
5. On the fifth operation you choose $p = 3$ and replace b with $[\,1, 1, 0, \underline{0}, 0\,]$.

Hence, sequence b changes in the following way: $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=0} [\,\underline{0}, 0\,] \xrightarrow{p=2} [\,1, 1, \underline{0}\,] \xrightarrow{p=1} [\,0, \underline{0}, 1, 0\,] \xrightarrow{p=3} [\,1, 1, 0, \underline{0}, 0\,]$. In the end the sequence b is equal to the sequence a, so this way to perform operations is one of the correct answers.

In the second test case, $n = 1$ and the only achieveable sequence b is $[\,0\,]$.

In the third test case, there are six possible sequences of operations:

1. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=0} [\,\underline{0}, 0\,] \xrightarrow{p=0} [\,\underline{0}, 0, 0\,]$.
2. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=0} [\,\underline{0}, 0\,] \xrightarrow{p=1} [\,1, \underline{0}, 0\,]$.
3. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=0} [\,\underline{0}, 0\,] \xrightarrow{p=2} [\,1, 1, \underline{0}\,]$.
4. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=1} [\,1, \underline{0}\,] \xrightarrow{p=0} [\,\underline{0}, 1, 0\,]$.
5. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=1} [\,1, \underline{0}\,] \xrightarrow{p=1} [\,0, \underline{0}, 0\,]$.
6. $[\,] \xrightarrow{p=0} [\,\underline{0}\,] \xrightarrow{p=1} [\,1, \underline{0}\,] \xrightarrow{p=2} [\,0, 1, \underline{0}\,]$.

None of them makes b equal to $[0, 1, 1]$, so the answer is "NO".

D. Ball Sorting

2 seconds, 256 megabytes

There are n colorful balls arranged in a row. The balls are painted in n distinct colors, denoted by numbers from 1 to n. The i-th ball from the left is painted in color c_i. You want to reorder the balls so that the i-th ball from the left has color i. Additionally, you have $k \geq 1$ balls of color 0 that you can use in the reordering process.

Due to the strange properties of the balls, they can be reordered only by performing the following operations:

1. Place a ball of color 0 anywhere in the sequence (between any two consecutive balls, before the leftmost ball or after the rightmost ball) while keeping the relative order of other balls. You can perform this operation no more than k times, because you have only k balls of color 0.
2. Choose any ball of **non-zero** color such that at least one of the balls adjacent to him has color 0, and move that ball (of non-zero color) anywhere in the sequence (between any two consecutive balls, before the leftmost ball or after the rightmost ball) while keeping the relative order of other balls. You can perform this operation as many times as you want, but for each operation you should pay 1 coin.

You can perform these operations in any order. After the last operation, all balls of color 0 magically disappear, leaving a sequence of n balls of non-zero colors.

What is the minimum amount of coins you should spend on the operations of the second type, so that the i-th ball from the left has color i for all i from 1 to n **after the disappearance of all balls of color zero**? It can be shown that under the constraints of the problem, it is always possible to reorder the balls in the required way.

Solve the problem for all k from 1 to n.

Input

The first line contains integer t ($1 \leq t \leq 500$) — the number of test cases. The descriptions of the test cases follow.

The first line contains one integer n ($1 \leq n \leq 500$) — the number of balls.

The second line contains n **distinct** integers $c_1, c_2, \ldots, c_n$ ($1 \leq c_i \leq n$) — the colors of balls from left to right.

It is guaranteed that sum of n over all test cases doesn't exceed 500.

Output

For each test case, output n integers: the i-th ($1 \leq i \leq n$) of them should be equal to the minimum amount of coins you need to spend in order to reorder balls in the required way for $k = i$.

input
3 6 2 3 1 4 6 5 3 1 2 3 11 7 3 4 6 8 9 10 2 5 11 1

output
3 2 2 2 2 2 0 0 0 10 5 4 4 4 4 4 4 4 4 4

In the first test case there are $n = 6$ balls. The colors of the balls from left to right are $[2, 3, 1, 4, 6, 5]$.

Let's suppose $k = 1$. One of the ways to reorder the balls in the required way for 3 coins:

$$[2, 3, 1, 4, 6, 5] \xrightarrow{1} [2, 3, 1, 4, 0, 6, 5] \xrightarrow{2} [2, 3, 4, 1, 0, 6, 5] \xrightarrow{2} [1, 2, 3, 4, 0, 6, 5] \xrightarrow{2} [1, 2, 3, 4, 0, 5, 6]$$

The number above the arrow is the operation type. Balls inserted on the operations of the first type are highlighted red; balls moved on the operations of second type are highlighted blue.

It can be shown that for $k = 1$ it is impossible to rearrange balls in correct order for less than 3 coins.

Let's suppose $k = 2$. One of the ways to reorder the balls in the required way for 2 coins:

$$[2, 3, 1, 4, 6, 5] \xrightarrow{1} [2, 3, 1, 4, 6, 0, 5] \xrightarrow{2} [2, 3, 1, 4, 0, 5, 6] \xrightarrow{1} [2, 3, 0, 1, 4, 0, 5, 6] \xrightarrow{2} [1, 2, 3, 0, 4, 0, 5, 6]$$

Note that this sequence of operations is also correct for k greater than 2.

It can be shown that for k from 2 to 6 it is impossible to rearrange balls in correct order for less than 2 coins.

In the second test case the balls are already placed in the correct order, so answers for all k are equal to 0.

E. Decreasing Game

2 seconds, 256 megabytes

This is an interactive problem.

Consider the following game for two players:

- Initially, an array of integers $a_1, a_2, \ldots, a_n$ of length n is written on blackboard.
- Game consists of *rounds*. On each round, the following happens:
  - The first player selects any i such that $a_i > 0$. If there is no such i, the first player loses the game (the second player wins) and game ends.
  - The second player selects any $j \neq i$ such that $a_j > 0$. If there is no such j, the second player loses the game (the first player wins) and game ends.
  - Let $d = \min(a_i, a_j)$. The values of a_i and a_j are simultaneously decreased by d and the next round starts.

It can be shown that game always ends after the finite number of rounds.

You have to select which player you will play for (first or second) and win the game.

Input

The first line contains a single integer n ($1 \leq n \leq 300$) — the length of array a.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 300$) — array a.

Interaction

Interaction begins after reading n and array a.

You should start interaction by printing a single line, containing either "First" or "Second", representing the player you select.

On each round, the following happens:

- If you are playing as the first player, you should print a single integer i ($1 \leq i \leq n$) on a separate line. After that, you should read a single integer j ($-1 \leq j \leq n$) written on a separate line.
  If $j = -1$, then you made an incorrect move. In this case your program should terminate immediately.

  If $j = 0$, then the second player can't make a correct move and you win the game. In this case your program should also terminate immediately.

Otherwise j is equal to the index chosen by the second player, and you should proceed to the next round.

- If you are playing as the second player, you should read a single integer i ($-1 \le i \le n$) written on a separate line.
  If $i = -1$, then you made an incorrect move on the previous round (this cannot happen on the first round). In that case your program should terminate immediately.

  If $i = 0$, then the first player can't make a correct move and you win the game. In this case your program should also terminate immediately.

  Otherwise i is equal to the index chosen by first player. In this case you should write single integer j ($1 \le j \le n$) on a separate line and proceed to the next round.

After printing i or j, do not forget to output the end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see the documentation for other languages.

Hacks

Hacks are disabled in this problem.

input

```
4
10 4 6 3


3

1

0
```

output

```
First
1

2

4
```

input

```
6
4 5 5 11 3 2

2

5

4

6

1

0
```

output

```
Second

4

4

3

1

3
```

In the first example $n = 4$ and array a is $[\,10, 4, 6, 3\,]$. The game goes as follows:

- After reading array a contestant's program chooses to play as the first player and prints "`First`".
- First round: the first player chooses $i = 1$, the second player chooses $j = 3$. $d = \min(a_1, a_3) = \min(10, 6) = 6$ is calculated. Elements a_1 and a_3 are decreased by 6. Array a becomes equal to $[\,4, 4, 0, 3\,]$.
- Second round: the first player chooses $i = 2$, the second player chooses $j = 1$. $d = \min(a_2, a_1) = \min(4, 4) = 4$ is calculated. Elements a_2 and a_1 are decreased by 4. Array a becomes equal to $[\,0, 0, 0, 3\,]$.
- Third round: the first player chooses $i = 4$. There is no $j \ne 4$ such that $a_j > 0$, so the second player can't make a correct move and the first player wins. Jury's program prints $j = 0$. After reading it, contestant's program terminates.

In the second example $n = 6$ and array a is $[\,4, 5, 5, 11, 3, 2\,]$. The game goes as follows:

- Contestant's program chooses to play as the second player and prints "`Second`".
- First round: $i = 2$, $j = 4$, $a = [\,4, 0, 5, 6, 3, 2\,]$.
- Second round: $i = 5$, $j = 4$, $a = [\,4, 0, 5, 3, 0, 2\,]$.
- Third round: $i = 4$, $j = 3$, $a = [\,4, 0, 2, 0, 0, 2\,]$.
- Fourth round: $i = 6$, $j = 1$, $a = [\,2, 0, 2, 0, 0, 0\,]$.
- Fifth round: $i = 1$, $j = 3$, $a = [\,0, 0, 0, 0, 0, 0\,]$.
- Sixth round: the first player can't make a correct move and the second player wins. Jury's program prints $i = 0$. After reading it, contestant's program terminates.

Note that the example interaction contains extra empty lines so that it's easier to read. The real interaction doesn't contain any empty lines and you shouldn't print any extra empty lines as well.