Codeforces β
Sponsored by Telegram

Kotlin Heroes: Practice 4

A. A+B (Trial Problem)

time limit per test: 2.0 s
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given two integers a and b. Print $a + b$.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases in the input. Then t test cases follow.

Each test case is given as a line of two integers a and b ($-1000 \le a, b \le 1000$).

Output

Print t integers — the required numbers $a + b$.

Example

input

```
4
1 5
314 15
-99 99
123 987
```

output

```
6
329
0
1110
```

B. Square?

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

Vasya claims that he had a paper square. He cut it into two rectangular parts using one vertical or horizontal cut. Then Vasya informed you the dimensions of these two rectangular parts. You need to check whether Vasya originally had a square. In other words, check if it is possible to make a square using two given rectangles.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases in the input. Then t test cases follow.

Each test case is given in two lines.

The first line contains two integers a_1 and b_1 ($1 \le a_1, b_1 \le 100$) — the dimensions of the first one obtained after cutting rectangle. The sizes are given in random order (that is, it is not known which of the numbers is the width, and which of the numbers is the length).

The second line contains two integers a_2 and b_2 ($1 \le a_2, b_2 \le 100$) — the dimensions of the second obtained after cutting rectangle. The sizes are given in random order (that is, it is not known which of the numbers is the width, and which of the numbers is the length).

Output

Print t answers, each of which is a string "YES" (in the case of a positive answer) or "NO" (in the case of a negative answer). The letters in words can be printed in any case (upper or lower).

Example

input

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

output

```
Yes
Yes
No
```

C. Sum of Round Numbers

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

A positive (strictly greater than zero) integer is called *round* if it is of the form `d00...0`. In other words, a positive integer is round if all its digits except the leftmost (most significant) are equal to zero. In particular, all numbers from 1 to 9 (inclusive) are round.

For example, the following numbers are round: 4000, 1, 9, 800, 90. The following numbers are **not** round: 110, 707, 222, 1001.

You are given a positive integer n ($1 \le n \le 10^4$). Represent the number n as a sum of round numbers using the minimum number of summands (addends). In other words, you need to represent the given number n as a sum of the least number of terms, each of which is a round number.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases in the input. Then t test cases follow.

Each test case is a line containing an integer n ($1 \le n \le 10^4$).

Output

Print t answers to the test cases. Each answer must begin with an integer k — the minimum number of summands. Next, k terms must follow, each of which is a round number, and their sum is n. The terms can be printed in any order. If there are several answers, print any of them.

Example

input

```
5
5009
7
9876
10000
10
```

output

```
2
5000 9
1
7
4
800 70 6 9000
1
10000
1
10
```

D. Alice, Bob and Candies

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

There are n candies in a row, they are numbered from left to right from 1 to n. The size of the i-th candy is a_i.

Alice and Bob play an interesting and tasty game: they eat candy. Alice will eat candy **from left to right**, and Bob — **from right to left**. The game ends if all the candies are eaten.

The process consists of moves. During a move, the player eats one or more sweets from her/his side (Alice eats from the left, Bob — from the right).

Alice makes the first move. During the first move, she will eat 1 candy (its size is a_1). Then, each successive move the players alternate — that is, Bob makes the second move, then Alice, then again Bob and so on.

On each move, a player counts the total size of candies eaten during the current move. Once this number becomes strictly greater than the total size of candies eaten by the other player on their previous move, the current player stops eating and the move ends. In other words, on a move, a player eats the smallest possible number of candies such that the sum of the sizes of candies eaten on this move is **strictly greater** than the sum of the sizes of candies that the other player ate on the **previous** move. If there are not enough candies to make a move this way, then the player eats up all the remaining candies and the game ends.

For example, if $n = 11$ and $a = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]$, then:

- move 1: Alice eats one candy of size 3 and the sequence of candies becomes $[1, 4, 1, 5, 9, 2, 6, 5, 3, 5]$.
- move 2: Alice ate 3 on the previous move, which means Bob must eat 4 or more. Bob eats one candy of size 5 and the sequence

of candies becomes $[1, 4, 1, 5, 9, 2, 6, 5, 3]$.
- move 3: Bob ate 5 on the previous move, which means Alice must eat 6 or more. Alice eats three candies with the total size of $1 + 4 + 1 = 6$ and the sequence of candies becomes $[5, 9, 2, 6, 5, 3]$.
- move 4: Alice ate 6 on the previous move, which means Bob must eat 7 or more. Bob eats two candies with the total size of $3 + 5 = 8$ and the sequence of candies becomes $[5, 9, 2, 6]$.
- move 5: Bob ate 8 on the previous move, which means Alice must eat 9 or more. Alice eats two candies with the total size of $5 + 9 = 14$ and the sequence of candies becomes $[2, 6]$.
- move 6 (the last): Alice ate 14 on the previous move, which means Bob must eat 15 or more. It is impossible, so Bob eats the two remaining candies and the game ends.

Print the number of moves in the game and two numbers:

- a — the total size of all sweets eaten by Alice during the game;
- b — the total size of all sweets eaten by Bob during the game.

Input

The first line contains an integer t ($1 \le t \le 5000$) — the number of test cases in the input. The following are descriptions of the t test cases.

Each test case consists of two lines. The first line contains an integer n ($1 \le n \le 1000$) — the number of candies. The second line contains a sequence of integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 1000$) — the sizes of candies in the order they are arranged from left to right.

It is guaranteed that the sum of the values of n for all sets of input data in a test does not exceed $2 \cdot 10^5$.

Output

For each set of input data print three integers — the number of moves in the game and the required values a and b.

Example

input

```
7
11
3 1 4 1 5 9 2 6 5 3 5
1
1000
3
1 1 1
13
1 2 3 4 5 6 7 8 9 10 11 12 13
2
2 1
6
1 1 1 1 1 1
7
1 1 1 1 1 1 1
```

output

```
6 23 21
1 1000 0
2 1 2
6 45 46
2 2 1
3 4 2
4 4 3
```

E. Special Permutation

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

A permutation of length n is an array $p = [p_1, p_2, \dots, p_n]$, which contains every integer from 1 to n (inclusive) and, moreover, each number appears exactly once. For example, $p = [3, 1, 4, 2, 5]$ is a permutation of length 5.

For a given number n ($n \ge 2$), find a permutation p in which absolute difference (that is, the absolute value of difference) of any two neighboring (adjacent) elements is between 2 and 4, inclusive. Formally, find such permutation p that $2 \le |p_i - p_{i+1}| \le 4$ for each i ($1 \le i < n$).

Print any such permutation for the given integer n or determine that it does not exist.

Input

The first line contains an integer t ($1 \le t \le 100$) — the number of test cases in the input. Then t test cases follow.

Each test case is described by a single line containing an integer n ($2 \le n \le 1000$).

Output

Print t lines. Print a permutation that meets the given requirements. If there are several such permutations, then print any of them. If no such permutation exists, print -1.

Example

input

```
6
10
2
4
6
7
13
```

output

```
9 6 10 8 4 7 3 1 5 2
-1
3 1 4 2
5 3 6 2 4 1
5 1 3 6 2 4 7
13 9 7 11 8 4 1 3 5 2 6 10 12
```