Codeforces Round #640 (Div. 4)

A. 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

B. Same Parity Summands

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

You are given two positive integers n ($1 \le n \le 10^9$) and k ($1 \le k \le 100$). Represent the number n as the sum of k positive integers of the same parity (have the same remainder when divided by 2).

In other words, find $a_1, a_2, \ldots, a_k$ such that all $a_i > 0$, $n = a_1 + a_2 + \ldots + a_k$ and either all a_i are even or all a_i are odd at the same time.

If such a representation does not exist, then report it.

Input
The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases in the input. Next, t test cases are given, one per line.

Each test case is two positive integers n ($1 \le n \le 10^9$) and k ($1 \le k \le 100$).

Output
For each test case print:

- YES and the required values a_i, if the answer exists (if there are several answers, print any of them);

- NO if the answer does not exist.

The letters in the words YES and NO can be printed in any case.

Example

input
8 10 3 100 4 8 7 97 2 8 8 3 10 5 3 1000000000 9
output
YES 4 2 4 YES 55 5 5 35 NO NO YES 1 1 1 1 1 1 1 NO YES 3 1 1 YES 111111110 111111110 111111110 111111110 111111110 111111110 111111110 111111110 111111120

C. K-th Not Divisible by n

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

You are given two positive integers n and k. Print the k-th positive integer that is not divisible by n.

For example, if $n = 3$, and $k = 7$, then all numbers that are not divisible by 3 are: $1, 2, 4, 5, 7, 8, 10, 11, 13 \ldots$. The 7-th number among them is 10.

Input

The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases in the input. Next, t test cases are given, one per line.

Each test case is two positive integers n ($2 \le n \le 10^9$) and k ($1 \le k \le 10^9$).

Output

For each test case print the k-th positive integer that is not divisible by n.

Example

input
6 3 7 4 12 2 1000000000 7 97 1000000000 1000000000 2 1
output
10 15 1999999999 113 1000000001 1

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 $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 1000$) — 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 Elements

time limit per test: 1 second
memory limit per test: 64 megabytes
input: standard input
output: standard output

Pay attention to the non-standard memory limit in this problem.

In order to cut off efficient solutions from inefficient ones in this problem, the time limit is rather strict. Prefer to use compiled statically typed languages (e.g. C++). If you use Python, then submit solutions on PyPy. Try to write an efficient solution.

The array $a = [a_1, a_2, \ldots, a_n]$ ($1 \le a_i \le n$) is given. Its element a_i is called special if there exists a pair of indices l and r ($1 \le l < r \le n$) such that $a_i = a_l + a_{l+1} + \ldots + a_r$. In other words, an element is called special if it can be represented as the sum of **two or more consecutive elements** of an array (no matter if they are special or not).

Print the number of special elements of the given array a.

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

- $a_3 = 4$ is a special element, since $a_3 = 4 = a_1 + a_2 = 3 + 1$;
- $a_5 = 5$ is a special element, since $a_5 = 5 = a_2 + a_3 = 1 + 4$;
- $a_6 = 9$ is a special element, since $a_6 = 9 = a_1 + a_2 + a_3 + a_4 = 3 + 1 + 4 + 1$;
- $a_8 = 6$ is a special element, since $a_8 = 6 = a_2 + a_3 + a_4 = 1 + 4 + 1$;
- $a_9 = 5$ is a special element, since $a_9 = 5 = a_2 + a_3 = 1 + 4$.

Please note that some of the elements of the array a may be equal — if several elements are equal and special, then all of them should be counted in the answer.

Input
The first line contains an integer t ($1 \le t \le 1000$) — 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 an integer n ($1 \le n \le 8000$) — the length of the array a. The second line contains integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$).

It is guaranteed that the sum of the values of n for all test cases in the input does not exceed 8000.

Output
Print t numbers — the number of special elements for each of the given arrays.

Example

input
5 9 3 1 4 1 5 9 2 6 5 3 1 1 2 5 1 1 1 1 1 8 8 7 6 5 4 3 2 1 1 1
output
5 1 0 4 0

F. Binary String Reconstruction

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

For some binary string s (i.e. each character s_i is either '0' or '1'), all pairs of consecutive (adjacent) characters were written. In other words, all substrings of length 2 were written. For each pair (substring of length 2), the number of '1' (ones) in it was calculated.

You are given three numbers:

- n_0 — the number of such pairs of consecutive characters (substrings) where the number of ones equals 0;
- n_1 — the number of such pairs of consecutive characters (substrings) where the number of ones equals 1;
- n_2 — the number of such pairs of consecutive characters (substrings) where the number of ones equals 2.

For example, for the string $s =$"1110011110", the following substrings would be written: "11", "11", "10", "00", "01", "11", "11", "11", "10". Thus, $n_0 = 1$, $n_1 = 3$, $n_2 = 5$.

Your task is to restore **any** suitable binary string s from the given values n_0, n_1, n_2. It is guaranteed that at least one of the numbers n_0, n_1, n_2 is greater than 0. Also, it is guaranteed that a solution exists.

Input
The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases in the input. Then test cases follow.

Each test case consists of one line which contains three integers n_0, n_1, n_2 ($0 \le n_0, n_1, n_2 \le 100$; $n_0 + n_1 + n_2 > 0$). It is

guaranteed that the answer for given n_0, n_1, n_2 exists.

Output
Print t lines. Each of the lines should contain a binary string corresponding to a test case. If there are several possible solutions, print any of them.

Example

input
7 1 3 5 1 1 1 3 9 3 0 1 0 3 1 2 0 0 3 2 0 0
output
1110011110 0011 0110001100101011 10 0000111 1111 000

G. 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 $p = [p_1, p_2, \ldots, p_n]$, 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 ($n \geq 2$), 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 $2 \leq |p_i - p_{i+1}| \leq 4$ for each i ($1 \leq i < n$).

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

Input
The first line contains an integer t ($1 \leq t \leq 100$) — 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 ($2 \leq n \leq 1000$).

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