Problem A: Deficient, Perfect, and Abundant

Input file: dpa.in
Output file: dpa.out

Write a program that repeatedly reads a positive integer, determines if the integer is *deficient*, *perfect*, or *abundant*, and outputs the number along with its classification.

A positive integer, *n*, is said to be *perfect* if the sum of its proper divisors equals the number itself. (Proper divisors include 1 but not the number itself.) If this sum is less that *n*, the number is *deficient*, and if the sum is greater than *n*, the number is *abundant*.

The input starts with the number of integers that follow. For each of the following integers, your program should output the classification, as given below. You may assume that the input integers are greater than 1 and less than 32500.

Sample input

```
3
4
6
12
```

Sample output

```
4 is a deficient number.

6 is a perfect number.

12 is an abundant number.
```

Problem B: Divisibility by 11

Input file: `div.in`
Output file: `div.out`

Write a program which accepts as input a positive integer and checks, using the algorithm described below, to see whether or not the integer is divisible by 11. This particular test for divisibility by 11 was given in 1897 by Charles L. Dodgson (Lewis Carroll).

Algorithm:

As long as the number being tested has more than two digits, form a new number by:

- deleting the units digit
- subtracting the deleted digit from the shortened number

The remaining number is divisible by 11 if and only if the original number is divisible by 11.

Note:

Leading zeroes are not considered part of the number and should not be printed.

As usual, the first number in the input indicates the number of positive integers that follow. Each positive integer has a maximum of 50 digits. You may assume no leading zeroes exist in the positive integers.

For each positive integer in the input, the output consists of a series of numbers formed as a digit is deleted and subtracted, followed by a message indicating whether or not the original number is divisible by 11. Outputs for different positive integers are separated by blank lines.

Sample input

```
1
12345678901234567900
```

Sample output

```
12345678901234567900
1234567890123456790
123456789012345679
12345678901234558
1234567890123447
123456789012337
12345678901226
1234567890116
123456789005
12345678995
1234567884
123456784
12345674
1234563
123453
12342
```

```
1232
121
11
The number 12345678901234567900 is divisible by 11.
```

Problem C: Pattern Generator

Input file: pat.in
Output file: pat.out

Write a program that repeatedly reads two numbers *n* and *k* and prints all bit patterns of length *n* with *k* ones in descending order (when the bit patterns are considered as binary numbers). You may assume that 30 >= *n* > 0, 8 > *k* >= 0, and *n* >= *k*. The first number in the input gives the number of pairs *n* and *k*. The numbers *n* and *k* are separated by a single space. Leading zeroes in a bit pattern should be included. See the example below.

Sample input

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

Sample output

```
The bit patterns are

10
01

The bit patterns are
00

The bit patterns are
1100
1010
1001
0110
0101
0011
```

Problem D: When in Rome...

Input file: `rom.in`
Output file: `rom.out`

If the Roman Empire had not fallen, then Rome would surely have discovered electricity and used electronic calculators; however, the Romans used Roman Numerals! Your task is to implement a simple Roman Calculator which accepts two Roman Numerals and outputs the sum in Roman Numerals. You may assume that numbers greater than 1000 will not occur in the input. Output numbers greater than 1000 are illegal and should generate the message `CONCORDIA CUM VERITATE` (In Harmony with Truth).

The input consists of a number, indicating the number of test cases, followed by this many test cases. Each test case consists of a single line with two numbers in Roman Numerals separated by a `+` along with an `=` at the end. There are no separating spaces.

For each test case the output is a copy of the input with the Roman Numeral that represents the sum. Outputs for different test cases are separated by a blank line.

Roman Research

The Roman Numerals used by the Romans evolved over many years, and so there are some variations in the way they are written. We will use the following definitions:

1. The following symbols are used: `I` for 1, `V` for 5, `X` for 10, `L` for 50, `C` for 100, `D` for 500, and `M` for 1000.
2. Numbers are formed by writing symbols from 1. from left to right, as a sum, each time using the symbol for the largest possible value. The symbols `M`, `C`, `X`, or `I` may be used at most three times in succession. Only if this rule would be violated, you can use the following rule:
    - When a single `I` immediately precedes a `V` or `X`, it is subtracted. When a single `X` immediately precedes an `L` or `C`, it is subtracted. When a single `C` immediately precedes a `D` or `M`, it is subtracted.

For example: `II` = 2; `IX` = 9; `CXIII` = 113; `LIV` = 54; `XXXVIII` = 38; `XCIX` = 99.

Sample input

```
3
VII+II=
XXIX+X=
M+I=
```

Sample output

```
VII+II=IX
XXIX+X=XXXIX
M+I=CONCORDIA CUM VERITATE
```

Problem E: Maximum Distance

Input file: max.in
Output file: max.out

Consider two descending sequences of integers X[0..*n*-1] and Y[0..*n*-1] with X[*i*] >= X[*i*+1] and Y[*i*] >= Y[*i*+1] and for all *i*, 0 <= *i* < *n* - 1. The *distance* between two elements X[*i*] and Y[*j*] is given by

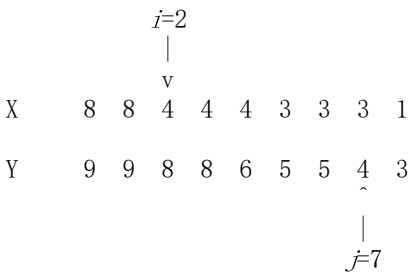
d(X[*i*], Y[*j*]) = *j* - *i* if *j* >= *i* and Y[*j*] >= X[*i*], or 0 otherwise

The distance between sequence X and sequence Y is defined by

d(X, Y) = max{d(X[*i*], Y[*j*]) | 0 <= *i* < *n*, 0 <= *j* < *n*}

You may assume 0 < *n* < 1000.

For example, for the sequences X and Y below, their maximum distance is reached for *i*=2 and *j*=7, so d(X, Y)=d(X[2], Y[7])=5.



Part (a)

There is a maximum value of d(X, Y) over all sequences X and Y of length *n*. What property must the sequences satisfy in order to reach this value? There is a minimum value of d(X, Y). What property must the sequences satisfy in order to reach this value?

Part (b)

Write a program that repeatedly reads a pair of sequences of integers and prints the distance between those sequences. The first sequence is the X sequence and the second is the Y sequence. You may assume that the sequences are descending and of equal length. A pair of sequences is preceded by a number on a single line indicating the number of elements in the sequences. Numbers in a sequence are separated by a space, and each sequence is on a single line by itself. As usual, the first number in the input gives the number of test cases. Try to write an *efficient* program.

Part (c)

Give a very *brief* explanation of your program. Also, give a rough estimate of the maximum number of comparisons between elements of the two sequences that your program computes. (For example, *n*^2 can be considered a "rough estimate" of *n*^2 - 4.)

Sample input

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

Sample output

```
The maximum distance is 5

The maximum distance is 0
```