Problem A: Beautiful Strings
Source code file: beautiful.{c, cpp, java}
Input file: beautiful.in

This problem involves manipulating strings. First, some terms:

- Given strings S1 and S2, we say that S2 can be *derived from* S1 if and only if we can start with S1 and remove selected characters until S2 results. We cannot rearrange or alter characters in S1, only remove them.
- A string is *unique* if and only if each character in it appears only once in that string.
- Given strings S1 and S2, we say that S1 is more *beautiful* than S2 if and only if S1 is longer, or (if they're the same length) S1 is lexicographically later than S2. ("Lexicographically later" means appearing later in a case-sensitive dictionary-order sort.)

Your task is to write a program to answer the following question: Given a string S, what is the most beautiful unique string that can be derived from it?

Your input file begins with an integer N, specifying the number of strings in the input file. This is followed by N strings. Each string is at least 3 characters long, no more than 100,000 characters long, and made up entirely of printable non-whitespace ASCII characters (no embedded spaces, tabs, or newlines, but digits or punctuation may be present). Do not assume each string is on a line by itself.

Output for each case is the most beautiful unique string that can be derived for that case, on a line by itself.

Sample Input	Sample Output
5 abCDef ffaabbccddeeffgg9032: ffaabbccd:";deeffgg9032: Ca1aoip231adjlk IHaveADogHisNameIsFido	abCDef fabcdeg9032: fabcd:";eg9032 Caoip231djlk IHaveADogisNmFd

Problem B: The Coin Game
Source code file: coins.{c, cpp, java}
Input file: coins.in

Alice and Ben play a simple coin game, as follows: A collection of coins of various denominations are laid out in a row. Alice plays first, removing a coin from one of the ends. Then Ben removes a coin from one end of the remaining row, and so on. Coins can be removed from either end of the row, but only from the ends. This continues until all coins have been chosen. Each player's score is the value of all the coins they collected, minus the value of the coins the other player collected. The player with the higher total wins the game. Given a row of coins, what is the best score Alice can achieve if Ben plays as well as possible?

Note that a simple greedy strategy of selecting the largest coin available will not always work. For example, if the coins' value on Alice's turn are:

5 25 10 1

In this case, Alice should select the coin of value 1. Doing so ensures that she can take the coin worth 25 on her next turn. (Ben must take the 5, or the 10; either way, Alice can take the 25).

You are given several sets of input. Each begins with an integer N ($0 < N <= 150$) specifying how many coins are in that game. This is followed by the N values of the coins in that game. Values will not necessarily correspond to values used by U.S. (or any other) currency system, but will always be integers greater than 0 and less than 100. Input values are separated by whitespace; do not assume input is line oriented. The end of input is marked by a value of 0 for N.

You should assume that at each move of the game, each player will select the best move they can. Output for each case is a single integer: the expected amount by which the first player will win. If the first player will lose, this number will be less than 0.

Sample Input	Sample Output
4 5 25 10 1 4 3 9 12 15 2 36 36 3 5 25 5 0	11 9 0 -15

Problem C: Hogwarts Sorting Hat
Source code file: hogwarts.{c, cpp, java}
Input file: hogwarts.in

The Sorting Hat at Hogwarts School is in charge of deciding to which of the four houses, Gryffindor, Hufflepuff, Ravenclaw or Slytherin, a young wizard is assigned. The Sorting Hat is getting tired of assigning young wizards and has asked you to write a program to help it assign the names of some of the young wizards to one of the four houses.

The Sorting hat has provided the following information about each house.

Gryffindor House Ghost: Nearly Headless Nick Head of House: Minerva McGonagall Named After: Godric Gryffindor Key Characters: "G", "Y" and "O"	Hufflepuff House Ghost: The Fat Friar Head of House: Professor Sprout Named After: Helga Hufflepuff Key Characters: "H", "F" and "U"
Ravenclaw House Ghost: The Grey Lady Head of House: Prof. Flitwick Named After: Rowena Ravenclaw Key Characters: "R", "V" and "A"	Slytherin House Ghost: The Bloody Baron Head of House: Severus Snape Named After: Salazar Slytherin Key Characters: "S", "B" and "E"

The Sorting Hat has decided to determine the placement of young wizards according to the following rules: For each young wizard, count the total number of key characters that appear in that young wizard's name and place the young wizard into the house in which his/her name contains the highest number of key characters. When a tie occurs for the highest number of characters for a house or if a name does not contain any of the key characters, the Sorting Hat must decide which house the young wizard will be assigned. Repeated characters should be counted. Naturally, upper-case and lower-case characters are counted equally.

Write a program that implements the algorithm described above. Your input will begin with an integer N > 0, the number of wizards in this year's entering class. This will then be followed by N names, separated by whitespace. Each wizard has a first and last name. For each name, the output will either be the name of the house or the message, "The Sorting Hat Must Decide."

Sample Input	Sample Output
6 Suffie Sneab Rave Graymund Minni Nick Goddard Joyself Henry Flutie Harrietta Ufhog	Slytherin Ravenclaw The Sorting Hat Must Decide. Gryffindor Hufflepuff The Sorting Hat Must Decide.

Problem D: Combination Lock
Source code file: lock.{c, cpp, java}
Input file: lock.in

The combination lock of this problem consists of a circular dial, which can be turned (clockwise or counterclockwise) and is embedded into the "fixed" part of the lock. The dial has *N* evenly spaced "ticks". The ticks are numbered from 0 to *N*-1, increasing in the clockwise direction. The fixed part of the lock has a "mark" which always "points to" a particular tick on the dial. Of course, the mark points to different ticks as the dial is turned. The lock comes with three code numbers T_1,T_2, T_3. These are non-negative integers and each of them is less than *N*. No two of the three are the same.

The lock is opened in three stages of operations:

1. If the mark initially points to tick T_1, turn the dial exactly two full revolutions clockwise and stop. Otherwise, turn the dial clockwise exactly two full revolutions clockwise and continue to turn it clockwise until the mark points to tick T_1.
2. Turn the dial one full revolution counterclockwise and continue to turn it counterclockwise until the mark points to tick T_2.
3. Turn the dial clockwise until the mark points to tick T_3. The lock should now open.

Given the numbers *N*, T_1, T_2, T_3, the goal of this problem is to find the average number of ticks the dial must be turned in order to open the lock. For any particular *N*, T_1, T_2, T_3 and a particular initial configuration of the lock, the number of ticks turned is defined to be the sum of the ticks turned in the three stages outlined above.

Input

The input file consists of a number of test cases, one test case per line. Each line of the input file contains four integers: *N*, T_1, T_2, T_3, in this order, separated by blank spaces. The integer *N* is a multiple of 5, $25 \leq N \leq 100$. The numbers T_1, T_2 and T_3 satisfy the constraints stated under the description above. The input will be terminated by a line containing only four zeroes, separated by blank spaces.

Output

For each test case, print the average number of ticks (rounded to three decimal places) the dial must be turned in order to open the lock. Assume that each of the *N* possible initial configurations of the lock is equally likely.

Sample Input

```
80 20 40 50
80 10 79 12
0 0 0 0
```

Sample Output

```
369.500
415.500
```

Problem E: Prime Bases
Source code file: primebases.{c, cpp, java}
Input file: primebases.in

Given any integer base $b \geq 2$, it is well known that every positive integer n can be uniquely represented in base b. That is, we can write

$$n = a_0 + a_1*b + a_2*b*b + a_3*b*b*b + ...$$

where the coefficients a_0, a_1, a_2, a_3, ... are between 0 and b-1 (inclusive).

What is less well known is that if p_0, p_1, p_2, ... are the first primes (starting from 2, 3, 5, ...), every positive integer n can be represented uniquely in the "mixed" bases as:

$$n = a_0 + a_1*p_0 + a_2*p_0*p_1 + a_3*p_0*p_1*p_2 + ...$$

where each coefficient a_i is between 0 and p_i-1 (inclusive). Notice that, for example, a_3 is between 0 and p_3-1, even though p_3 may not be needed explicitly to represent the integer n.

Given a positive integer n, you are asked to write n in the representation above. Do not use more primes than it is needed to represent n, and omit all terms in which the coefficient is 0.

Input

Each line of input consists of a single positive 32-bit signed integer. The end of input is indicated by a line containing the integer 0.

Output

For each integer, print the integer, followed by a space, an equal sign, and a space, followed by the mixed base representation of the integer in the format shown below. The terms should be separated by a space, a plus sign, and a space. The output for each integer should appear on its own line.

Sample Input

```
123
456
123456
0
```

Sample Output

```
123 = 1 + 1*2 + 4*2*3*5
456 = 1*2*3 + 1*2*3*5 + 2*2*3*5*7
123456 = 1*2*3 + 6*2*3*5 + 4*2*3*5*7 + 1*2*3*5*7*11 + 4*2*3*5*7*11*13
```