Problem 1—Same Digits

Professor Plum likes to keep his elementary-aged children minds active over the summer months with simple mathematical puzzles. He asked them to answer the following puzzle.

Find the smallest positive integer x, such that 2x, 3x, 4x, 5x and 6x all contain exactly the same set of digits.

INPUT SPECIFICATION.

This problem needs no file input, or any other input.

OUTPUT SPECIFICATION.

This problem should produce only a single line of output containing the smallest such positive integer x.

Problem 2—Mountain Sort

Professor Plum likes to bicycle in the Rocky Mountains during his summer vacation. While writing an array question for his final examination in CS 101, he invents the notion of a *mountain sort* where the first half of the array is in ascending order and last half of the array is in descending order. More specifically, the smallest item is in the first index, the second smallest item is in the last index, the third smallest item is in the second index, the fourth smallest item is in the next to last index, etc.

For example, an array initially order as: 20, 45, 30, 5, 15, 50, 10, 30 would be mountain sorted to:

5, 15, 30, 45, 50, 30, 20, 10.

INPUT SPECIFICATION.

The first line of the input file contains an integer count of the number of items to mountain sort. The remaining lines will contain one integer per line.

OUTPUT SPECIFICATION.

The first line of the output file should contain an integer count of the number of items mountain sorted. The remaining lines will contain one integer per line in mountain-sorted order.

SAMPLE INPUT.

8<EOLN>

20<EOLN>

45<EOLN>

30<EOLN>

5<EOLN>

15<EOLN>

50<EOLN>

10<EOLN>

30<EOLN>

<EOF>

SAMPLE·OUTPUT.

8<EOLN>

5<EOLN>

15<EOLN>

30<EOLN>

45<EOLN>

50<EOLN>

30<EOLN>

20<EOLN>

10<EOLN>

<EOF>

Problem 3—IOU

Professor Plum likes working at UNI (University of Northern Iowa). With all the budget cuts due to reduced State support, he thinks a better name might be “IOU”. For a protest at the State capitol, he wants you to write a program to generate ASCII art printing “IOU” vertically for a sign. Since he does not know the dimensions of the sign, he wants your program to take as input a positive integer scaling factor.

|  |  |  |  |
| --- | --- | --- | --- |
| Scaling Factor | Letter Dimension  (# characters × # characters) | Line Width of Letters  (# characters) | Blank Lines  Between Letters |
| 1 | 3 × 3 | 1 | 1 |
| 2 | 6 × 6 | 2 | 2 |
| 3 | 9 × 9 | 3 | 3 |
| 20 | 60 × 60 | 20 | 20 |

A scaling factor of 1 would produce:

III

I

III

OOO

O O

OOO

U U

U U

UUU

A scaling factor of 2 would produce:

IIIIII

IIIIII

II

II

IIIIII

IIIIII

OOOOOO

OOOOOO

OO OO

OO OO

OOOOOO

OOOOOO

UU UU

UU UU

UU UU

UU UU

UUUUUU

UUUUUU

INPUT SPECIFICATION – File name “prob3.in”

The input file contains a single line with a positive integer scaling factor for the sign.

OUTPUT SPECIFICATION.

The output file should contain the ASCII art for the sign corresponding to the scaling factor specified by the input file.

SAMPLE INPUT.

4<EOLN>

<EOF>

SAMPLE·OUTPUT.

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

····IIII····<EOLN> 🡸NOTICE THE DOTS (‘·’) REPRESENT BLANK SPACES

····IIII····<EOLN>

····IIII····<EOLN>

····IIII····<EOLN>

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

IIIIIIIIIIII<EOLN>

<EOLN>

<EOLN>

<EOLN>

<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOO····OOOO<EOLN>

OOOO····OOOO<EOLN>

OOOO····OOOO<EOLN>

OOOO····OOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

OOOOOOOOOOOO<EOLN>

<EOLN>

<EOLN>

<EOLN>

<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUU····UUUU<EOLN>

UUUUUUUUUUUU<EOLN>

UUUUUUUUUUUU<EOLN>

UUUUUUUUUUUU<EOLN>

UUUUUUUUUUUU<EOLN>

<EOF>

Problem 4—Poker Hands

Professor Plum likes to play poker, so he is trying to teach a Computer Science colleague to play.

In poker, the cards are valued in the ascending order:  
2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

A poker hand consists of five cards and are ranked, from lowest to highest, in the following way:

* **High Card**: Highest value card.
* **One Pair**: Two cards of the same value.
* **Two Pairs**: Two different pairs.
* **Three of a Kind**: Three cards of the same value.
* **Straight**: All cards are consecutive values.
* **Flush**: All cards of the same suit.
* **Full House**: Three of a kind and a pair.
* **Four of a Kind**: Four cards of the same value.
* **Straight Flush**: All cards are consecutive values of the same suit.
* **Royal Flush**: Ten, Jack, Queen, King, Ace, in same suit.

If two players have the same ranked hands, then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players(‘T’ is 10, ‘J’ is Jack, ‘Q’ is Queen, ‘K’ is King, ‘A’ is Ace):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hand** |  | **Player 1** |  | **Player 2** |  | **Winner** |
| **1** |  | 5H 5C 6S 7S KD  Pair of Fives |  | 2C 3S 8S 8D TD  Pair of Eights |  | Player 2 |
| **2** |  | 5D 8C 9S JS AC  Highest card Ace |  | 2C 5C 7D 8S QH  Highest card Queen |  | Player 1 |
| **3** |  | 2D 9C AS AH AC  Three Aces |  | 3D 6D 7D TD QD  Flush with Diamonds |  | Player 2 |
| **4** |  | 4D 6S 9H QH QC  Pair of Queens Highest card Nine |  | 3D 6D 7H QD QS  Pair of Queens Highest card Seven |  | Player 1 |
| **5** |  | 2H 2D 4C 4D 4S  Full House With Three Fours |  | 3C 3D 3S 9S 9D  Full House With Three Threes |  | Player 1 |

Professor Plum has a program for generating a file containing a specified number of random poker hands for two poker players. He plans on emailing this file to his colleague in training daily. He wants you to write a program to take this file and report the number of wins for Player 1 and Player 2.

INPUT SPECIFICATION – File name “prob4.in”

The first line of the input file contains an integer count of the number of lines in the rest of the file. Each of the remaining lines describe 10 cards with the first 5 being Player 1’s hand and second 5 being Player 2’s hand. Each card is represented by two characters as in the above example. Each of the 10 cards on a line is separated by a single blank space.

OUTPUT SPECIFICATION.

The output file should contain two lines of output. The first line should specify the number of hands won by player 1, and be formatted exactly as: “Player 1 won # hands.” with the ‘#’ replaced by the correct integer. The second line should specify the number of hands won by player 2, and be formatted exactly as: “Player 2 won # hands.” with the ‘#’ replaced by the correct integer.

SAMPLE INPUT.

6<EOLN>

8C TS KC 9H 4S 7D 2S 5D 3S AC<EOLN>

5C AD 5D AC 9C 7C 5H 8D TD KS<EOLN>

3H 7H 6S KC JS QH TD JC 2D 8S<EOLN>

TH 8H 5C QS TC 9H 4D JC KS JS<EOLN>

7C 5H KC QH JD AS KH 4C AD 4S<EOLN>

5H KS 9C 7D 9H 8D 3S 5D 5C AH<EOLN>

<EOF>

SAMPLE·OUTPUT.

Player 1 won 3 hands.<EOLN>

Player 2 won 3 hands.<EOLN>

<EOF>

Problem 5—A Million Combinations

Professor Plum likes to teach Discrete Mathematics and one of his favorite expressions is “There must be more than a million combinations”. Therefore, it is not too surprising that his favorite Combinatorics problem involves tallying the number of ways to achieve a million combinations.

Typically, he provides his students with the following background information.

“There are exactly ten ways of selecting three of five things. Consider the digits 12345, ten ways are:

123, 124, 125, 134, 135, 145, 234, 235, 245, and 345

In combinatorics, we use the notation, 5C3 = 10.

In general,

, where

Blaise Pascal (17th century mathematician) developed Pascal’s triangle to calculate the number of combinations in a “dynamic programming” fashion. Pascal’s triangle is written as:



It is not until *n* = 23, that a value exceeds one-million: 23C10 = 1144066.”

Given a specific integer value *max*, you are to determine how many <*n,r>* pairs (1 ≤ *r* ≤ *n* ≤ *max*) give values of  *n*C*r* greater than one-million?

The pairs do not necessarily need to give distinct *n*C*r* values. For example, pairs <23,10> and <23,13> (23C10 = 23C13 = 1144066) should each be tallied if *max* is 23 or bigger.

INPUT SPECIFICATION– File name “prob5.in”

The input file contains a single line with a positive integer *max*.

OUTPUT SPECIFICATION.

The output file should contain a single line with the number of <n,r> pairs (1 ≤ *r* ≤ *n* ≤ *max*) giving values of  *n*C*r* greater than one-million.

SAMPLE INPUT.

25<EOLN>

<EOF>

SAMPLE·OUTPUT.

21<EOLN>

<EOF>

Problem 6—Guessing Game

Professor Plum routinely drives a van-load of CS students to MICS. To keep awake, the “lucky” student in the front passenger seat gets to play the hidden-number guessing game with Professor Plum. They agree on the largest possible value *n* and the student selects a hidden number from the set of integers {1, 2, ..., *n*}. The professor repeatedly tries to guess the hidden number with the student responding with one of three possible answers:

* "Your guess is lower than the hidden number", or
* "Yes, that's it!", or
* "Your guess is higher than the hidden number".

On one MICS trip, a student got tired of Professor Plum doing a “binary search” of the range to minimize the number of guesses in the worst case. The student asked Professor Plum to consider a related optimization problem: “Given the value of *n*, what is the *optimal strategy* **that minimizes the sum of all incorrect guesses** leading up to the hidden number in the worst possible case.”

At first Professor Plum did not understand, so the student talked through some examples.

If *n*=3, the best we can do is obviously to guess the number "**2**". The answer will immediately lead us to find the hidden number (at a total cost = 2). The total cost is 2 because the one incorrect guess is 2, and not because we need 2 guesses.

If *n*=8, we might decide to use a "binary search" type of strategy: Our first guess would be "**4**" and in the worst case the hidden number is higher than 4, so we will need one or two additional guesses. Let our second guess be "**6**". If the hidden number is still higher than 6, we will need a third guess in order to discriminate between 7 and 8. Thus, our third guess will be "**7**" and the total cost for this worst-case scenario will be 4+6+7=**17**.

Professor Plum begins to understand when the student points out that we can improve considerably the worst-case cost for *n*=8, by guessing "**5**"first. If we are told that the hidden number is higher than 5, our second guess will be "**7**", then we'll know for certain what the hidden number is (for a total cost of 5+7=**12**). If we are told that the hidden number is lower than 5, our second guess will be "**3**" and if the hidden number is lower than 3 our third guess will be "**1**", giving a total cost of 5+3+1=**9**. Since **12**>**9**, the worst-case cost for this strategy is **12**. That's better than what we achieved previously with the "binary search" strategy; it is also better than or equal to any other strategy. So, in fact, we have just described an optimal strategy for *n*=8.

Let C(*n*) be the worst-case cost achieved by an optimal strategy for *n*, as described above.  
Thus C(1) = 0, C(2) = 1, C(3) = 2, C(8) = 12, C(20) = 49, and

Professor Plum want you to write a program that takes as input a single positive integer X, and calculates

*INPUT SPECIFICATION– File name “prob6.in”*

The input file contains a single line with a positive integer X. You program should find

OUTPUT SPECIFICATION.

The output file should contain a single line with a single integer answer corresponding to the input X.

SAMPLE INPUT.

20<EOLN>

<EOF>

SAMPLE·OUTPUT.

402<EOLN>

<EOF>

Problem 7—Secret Message

Professor Plum’s wife enjoys reading “spy” novels. She likes to send him encoded messages. He hates it, but at least her encryption scheme is simple. She replaces each letter with the corresponding

letter from the other end of the alphabet. A's become Z's. B's become Y's, etc. Capital letters are replaced with capital letters. Lower-case letters are replaced with lower-case letters. All other characters are left unchanged.

Given an encoded message, you are to decode it.

INPUT SPECIFICATION– File name “prob7.in”

The input file contains the encoded message possibly split across multiple lines with each line naturally ended by an <EOLN>.

OUTPUT SPECIFICATION.

The output should contain the decoded message possibly split across multiple lines corresponding to the input file layout.

SAMPLE INPUT. (NOTE: THE DOTS (‘·’) REPRESENT BLANK SPACES)

Gsv·jfrxp·yildm·ulc·qfnkvw·levi·gsv·ozab·wlth.<EOLN>

@#=zyxwvutsrqponmlkjihgfEDCBA<EOLN>

SAMPLE·OUTPUT.

The·quick·brown·fox·jumped·over·the·lazy·dogs.<EOLN>

@#=abcdefghijklmnopqrstuVWXYZ<EOLN>

<EOF>