ACM Pacific NW Region Programming Contest
13 November 2004

Problem A: Mersenne Composite Numbers

One of the world-wide cooperative computing tasks is the “Grand Internet Mersenne Prime Search” — GIMPS — striving to find ever-larger prime numbers by examining a particular category of such numbers.

A Mersenne number is defined as a number of the form (2^p–1), where p is a prime number — a number divisible only by one and itself. (A number that can be divided by numbers other than itself and one are called “composite” numbers, and each of these can be uniquely represented by the prime numbers that can be multiplied together to generate the composite number — referred to as its prime factors.)

Initially it looks as though the Mersenne numbers are all primes.

Prime	Corresponding Mersenne Number
2	`4-1 = 3  —  prime`
3	`8-1 = 7  —  prime`
5	`32-1 = 31  —  prime`
7	`128-1 = 127  —  prime`

If, however, we are having a “Grand Internet” search, that must not be the case.

Where *k* is an input parameter, compute all the Mersenne *composite* numbers less than 2^k — where $k \leq 63$ (that is, it will fit in a 64-bit signed integer on the computer). In Java, the “long” data type is a signed 64-bit integer. Under gcc and g++ (C and C++ in the programming contest environment), the “long long” data type is a signed 64-bit integer.

Input (from file a.in)

Input is from file a. in. It contains a single number, without leading or trailing blanks, giving the value of *k*. As promised, $k \leq 63$.

Output (to stdout)

One line per Mersenne composite number giving first the prime factors (in increasing order) separate by asterisks, an equal sign, the Mersenne number itself, an equal sign, and then the explicit statement of the Mersenne number, as shown in the sample output. Use exactly this format. Note that all separating white space fields consist of one blank.

Sample input

```
31
```

Sample output

```
23 * 89 = 2047 = ( 2 ^ 11 ) - 1
47 * 178481 = 8388607 = ( 2 ^ 23 ) - 1
233 * 1103 * 2089 = 536870911 = ( 2 ^ 29 ) - 1
```

ACM Pacific NW Region Programming Contest
13 November 2004

Problem B: Zipper

Problem description

Given three strings, you are to determine whether the third string can be formed by combining the characters in the first two strings. The first two strings can be mixed arbitrarily, but each must stay in its original order.

For example, consider forming "`tcraete`" from "`cat`" and "`tree`":

String A: `cat`
String B: `tree`
String C: `tcraete`

As you can see, we can form the third string by alternating characters from the two strings. As a second example, consider forming "`catrtee`" from "`cat`" and "`tree`":

String A: `cat`
String B: `tree`
String C: `catrtee`

Finally, notice that it is impossible to form "`cttaree`" from "`cat`" and "`tree`".

The Input (from file b.in)
The first line of input file, b.in, contains a single positive integer from 1 through 1000. It represents the number of data sets to follow. The processing for each data set is identical. The data sets appear on the following lines, one data set per line.

For each data set, the line of input consists of three strings, separated by a single space. All strings are composed of upper and lower case letters only. The length of the third string is always the sum of the lengths of the first two strings. The first two strings will have lengths between 1 and 200 characters, inclusive.

The Output (to stdout)
For each data set, print:

```
Data set n: yes
```

if the third string can be formed from the first two, or

```
Data set n: no
```

if it cannot. Of course n should be replaced by the data set number. See the sample output below for an example.
Sample input

```
3
cat tree tcraete
cat tree catrtee
cat tree cttaree
```

Sample output

```
Data set 1: yes
Data set 2: yes
Data set 3: no
```

ACM Pacific NW Region Programming Contest
13 November 2004
Problem C: Lenny's Lucky Lotto Lists

Lenny likes to play the game of lotto. In the lotto game, he picks a list of *N* unique numbers in the range from `1` to *M*. If his list matches the list of numbers that are drawn, he wins the big prize.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each number in it is at least twice as large as the one before it. So, for example, if `N = 4` and `M = 10`, then the possible lucky lists Lenny could like are:

```
1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10
```

Thus Lenny has four lists from which to choose.

Your job, given *N* and *M*, is to determine from how many lucky lists Lenny can choose.

Input (from file c.in)

There will be multiple cases to consider from input file c.in. The first input will be a number `C` (`0 < C <= 50`) indicating how many cases with which you will deal. Following this number will be pairs of integers giving values for *N* and *M*, in that order. You are guaranteed that `1 <= N <= 10, 1 <= M <= 2000,` and `N <= M`. Each `N M` pair will occur on a line of its own. `N` and `M` will be separated by a single space.

Output (to stdout)

For each case display a line containing the case number (starting with 1 and increasing sequentially), the input values for `N` and `M`, and the number of lucky lists meeting Lenny's requirements. The desired format is illustrated in the sample shown below.

Sample Input

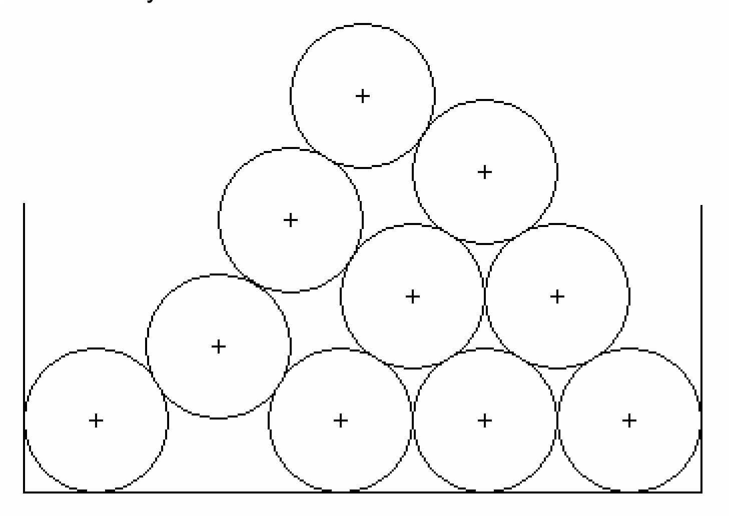
```
3
4 10
2 20
2 200
```

Sample Output

```
Case 1: n = 4, m = 10, # lists = 4
Case 2: n = 2, m = 20, # lists = 100
Case 3: n = 2, m = 200, # lists = 10000
```

ACM Pacific NW Region Programming Contest
13 November 2004
Problem D: Stacking Cylinders

Cylinders (e.g. oil drums) (of radius 1 foot) are stacked in a rectangular bin. Each cylinder on an upper row rests on two cylinders in the row below. The cylinders in the bottom row rest on the floor. Each row has one less cylinder than the row below.



This problem is to write a program to compute the location of the center of the top cylinder from the centers of the cylinders on the bottom row. Computations of intermediate values should use double precision.

The Input (from file d.in)

Each data set will appear in one line of the input in the file d.in. An input line consists of the number, ***n***, of cylinders on the bottom row followed by ***n*** floating point values giving the x coordinates of the centers of the cylinders (the y coordinates are all 1.0 since the cylinders are resting on the floor (y = 0.0)). The value of ***n*** will be between 1 and 10 (inclusive). The end of input is signaled by a value of ***n*** = 0. The distance between adjacent centers will be at least 2.0 (so the cylinders do not overlap) but no more than 3.4 (cylinders at level ***k*** will never touch cylinders at level ***k – 2***).

The Output (to stdout)

The output for each data set is a line containing the x coordinate of the topmost cylinder rounded to 4 decimal places, a space and the y coordinate of the topmost cylinder to 4 decimal places.
Note: To help you check your work, the x-coordinate of the center of the top cylinder should be the average of the x-coordinates of the leftmost and rightmost bottom cylinders.

Sample Input
```
4 1.0 4.4 7.8 11.2
1 1.0
6 1.0 3.0 5.0 7.0 9.0 11.0
10 1.0 3.0 5.0 7.0 9.0 11.0 13.0 15.0 17.0 20.4
5 1.0 4.4 7.8 14.6 11.2
0
```

Sample Output
```
6.1000 4.1607
1.0000 1.0000
6.0000 9.6603
10.7000 15.9100
7.8000 5.2143
```

ACM Pacific NW Region Programming Contest
13 November 2004
Problem E: Going Home

On a grid map there are *n* little men and *n* houses. In each unit time, every little man can move one unit step, either horizontally, or vertically, to an adjacent point. For each little man, you need to pay a $1 travel fee for every step he moves, until he enters a house. The task is complicated with the restriction that each house can accommodate only one little man.

Your task is to compute the minimum amount of money you need to pay in order to send these *n* little men into those *n* different houses. The input is a map of the scenario, a '.' means an empty space, an 'H' represents a house on that point, and am 'm' indicates there is a little man on that point.

You can think of each point on the grid map as a quite large square, so it can hold *n* little men at the same time; also, it is okay if a little man steps on a grid with a house without entering that house.

The Input (from file e.in)

There are one or more test cases in file e.in. Each case starts with a line giving two integers *N* and *M*, where *N* is the number of rows of the map, and *M* is the number of columns. The rest of the input will be *N* lines describing the map. You may assume both *N* and *M* are between 2 and 30, inclusive. There will be the same number of 'H's and 'm's on the map; and there will be at most 100 houses. Input will terminate with 0 0 for *N* and *M*.

The Output (to stdout)

For each test case, output one line with the single integer, which is the minimum amount, in dollars, you need to pay.

Sample Input

```
2 2
.m
H.
5 5
HH..m
.....
.....
.....
mm..H
7 8
...H....
...H....
...H....
mmmHmmmm
...H....
...H....
...H....
0 0
```

Sample Output

```
2
10
28
```

ACM Pacific NW Region Programming Contest
13 November 2004
Problem F: Specialized Four-Digit Numbers

Find and list all four-digit numbers in decimal notation that have the property that the sum of its four digits equals the sum of its digits when represented in hexadecimal (base 16) notation **and** also equals the sum of its digits when represented in duodecimal (base 12) notation.

For example, the number 2991 has the sum of (decimal) digits 2+9+9+1 = 21. Since 2991 = 1*1728 + 8*144 + 9*12 + 3, its duodecimal representation is 1893_{12}, and these digits also sum up to 21. But in hexadecimal 2991 is BAF_{16}, and 11+10+15 = 36, so 2991 should be rejected by your program.

The next number (2992), however, has digits that sum to 22 in all three representations (including $BB0_{16}$), so 2992 should be on the listed output. (We don't want decimal numbers with fewer than four digits — excluding leading zeroes — so that 2992 is the first correct answer.)

The Input (no input)

There is no input for this problem

The Output (to stdout)

Your output is to be 2992 and all larger four-digit numbers that satisfy the requirements (in strictly increasing order), each on a separate line with no leading or trailing blanks, ending with a new-line character. There are to be no blank lines in the output. The first few lines of the output are shown below.

Sample Input

There is no input for this problem

(partial) Sample Output

```
2992
2993
2994
2995
2996
2997
2998
2999
```

ACM Pacific NW Region Programming Contest
13 November 2004
Problem G: Jill's Tour Paths

Every year, Jill takes a bicycle tour between two villages. There are different routes she can take between these villages, but she does have an upper limit on the distance that she wants to travel. Given a map of the region indicating the cities and the roads between them (and their distances), Jill would like to have a list of the various routes between the selected cities that will meet her distance requirements. Your task is to write a program that will produce a list of these routes, in increasing order of distance.

We make the following assumptions.

- At most one road connects any pair of villages, and this road is two-way and has a non-zero positive distance.
- There are no roads that lead directly from a village back to the same village.
- Jill is only concerned about a one-way trip. That is, she is not concerned about returning to the village from which she starts her tour.
- Jill will not visit any village more than once during the tour.
- The farthest Jill will ever travel is 9999 units

The Input (from file g.in)

The input from file g.in will contain several possible cases, each including a route map, identification of the start and destination villages, and the maximum distance Jill is willing to travel.
Each case appears in the input as a set of integers separated by blanks and/or ends of lines. The order and interpretation of these integers is as follows:

- *NV* – the number of villages in the route map. This number will be no larger than 20.
- *NR* – the number of roads that appear in the route map. Each road connects a distinct pair of villages.
- *NR* triples, one for each road, containing *C1*, *C2*, and *DIST* – *C1* and *C2* identify two villages connected by a road, and *DIST* gives the distance between these villages on that road.
- *SV, DV* – the numbers associated with the start and destination villages; the villages are numbered 1 to *NV*.
- *MAXDIST* – the maximum distance Jill is willing to travel (one way).

The data for the last case will be followed by a single integer with the value –1.

The Output (to stdout)

For each case, display the case number (1, 2, …) on the first line of output. Then, each on a separate additional line, list the routes that Jill might take preceded by the length of the route. Order the routes first by length, from shortest to longest. Within routes having the same length, order them in increasing lexicographic order. The sample input and output provide suitable examples, and the formatting shown there should be followed closely (each village number should be separated by a single space).
Separate the output for consecutive cases by a single blank line.

Sample Input

```
4 5
1 2 2
1 3 3
1 4 1
2 3 2
3 4 4
1 3
4

4 5
1 2 2
1 3 3
1 4 1
2 3 2
3 4 4
1 4
10

5 7
1 2 2
1 4 5
2 3 1
2 4 2
2 5 3
3 4 3
3 5 2
1 3
8

-1
```

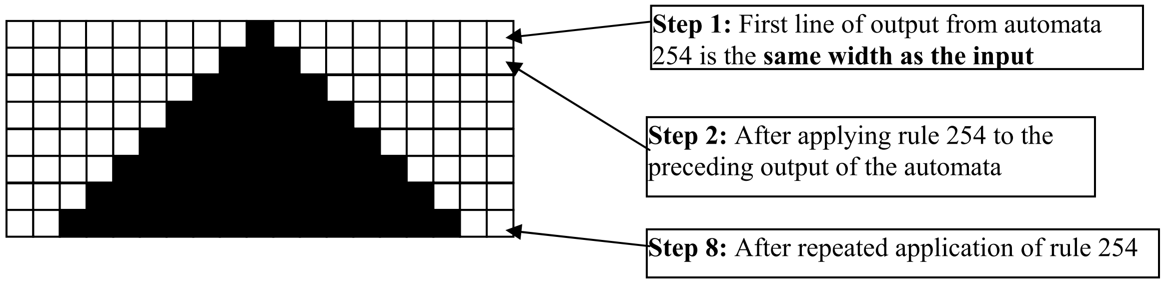
Sample Output

```
Case 1:
 3: 1 3
 4: 1 2 3

Case 2:
 1: 1 4
 7: 1 3 4
 8: 1 2 3 4

Case 3:
 3: 1 2 3
 7: 1 2 4 3
 7: 1 2 5 3
 8: 1 4 2 3
 8: 1 4 3
```

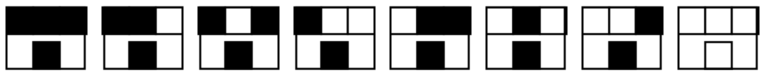
ACM Pacific NW Region Programming Contest
13 November 2004
Problem H: Boundaries on "A New Kind of Science"

Stephen Wolfram in a "New Kind of Science" describes a special kind of cellular automata. The automata he describes are rather interesting. They consist of rows of blocks, where blocks are either filled or not filled depending on the previous row. To generate a new row of blocks, the automata looks at the preceding row and then follows a pre-set "rule" to either color or not color a square on the output row.

For example the following diagram illustrates the "output" from one of these special kind of cellular automata:


Step 1: First line of output from automata 254 is the **same width as the input**
Step 2: After applying rule 254 to the preceding output of the automata
Step 8: After repeated application of rule 254

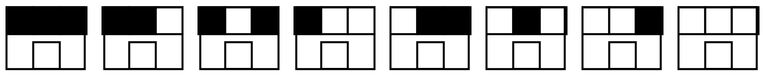
was generated by repeated application of "Rule 254". The automaton was initialized with an input line that consisted of a single black square. Repeated application of rule 254 to the ouput line from the preceding step generated the black triangle shown above.



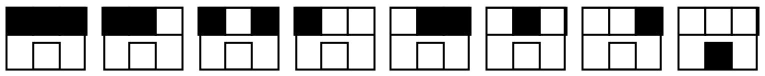
1 1 1 1 1 1 1 0 = 254 in base 10

For this rule, the top row in each box gives one of eight possible color combinations for a cell (the middle cell) and its two neighbors (the left and right neighbors of the cell). The bottom row in a box specifies the color that the center cell should have on the next step for each of the 8 possible cases.

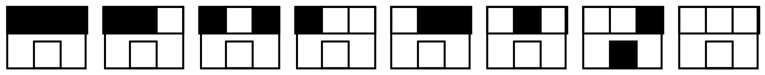
Given this arrangement, there are 255 different generating rules or automata numbered as follows:



0 0 0 0 0 0 0 0 = RULE 0 (base10)

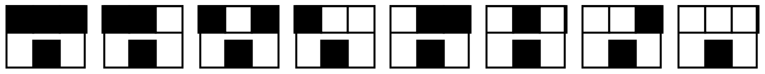


0 0 0 0 0 0 0 1 = RULE 1 (base10)



0 0 0 0 0 0 1 0 = RULE 2 (base10)

: : :



1 1 1 1 1 1 1 1 = RULE 255 (base10)

The "Bounded" Automata

- Unlike the automata in "A New Kind of Science", the automata for this problem operate in a "bounded space". **In other words, each line output by an automaton consists of exactly n squares**, and, the first square**,** and the last square of the output from a bounded automaton are always white no matter what the rule for the automata. This means that while an automaton can examine the first and last square of its input line when computing the new second and second to last characters, it cannot change the first or last square when it outputs a line. Thus, the **first and the last square on a line must always remain white**.
- Bounded automata will **always start life on a line with an odd number of squares**, and **all of the squares (except the middle square) are white**. The **middle square for step 1 is always black**.
- The number of squares is determined by the string for which the program is searching (the second field of an input line)

The Program

For **every line in the input file**, your program must determine which (if any) of the 255 possible automata could have generated that particular line when started from the standard starting state. If none of the automata generate the sequence by the given step number, output "NONE". If more than one automata generated that line, then, you must output all of the automata that generate the line as described below in the output section.

The Input

- The **first field** on a **line of input** consists of a **maximum step number** (**max_step**) for the automata to run. This number can be up to 32 bits. Values are chosen such that the problem is solvable within the given time limits given the input specifications.
- The **second field** on a line of input consists of an **odd-length string of characters** which represent the "squares" on the line. These will not be longer than 256 characters in length, and can be shorter. The **size** of the **second field** determines **n** -- the number of squares in a particular automata.
- The character '**W**' represents a **white square** and a '**B**' represents a **black square**.
- If an input string **contains characters other than 'W' or 'B'**, or **if the input string is an invalid string (not a properly bounded string as described previously)**, then obviously the string cannot be found by any of the automata, and the output will be **LINE # NONE** as illustrated in the sample output.
- **Each line in the input file will have a terminating newline character**.
- Input is terminated by a single line with the characters **"END OF INPUT"** as illustrated below. The **END OF INPUT** line should not be processed by your search algorithm.

The Output

- The output consists of **LINE #** followed by pairs of numbers **(rule, step)** where **rule** is the rule number of the automata **(0 through 255)** that generated a particular output and **step** is the **first step in the sequence of outputs (1, 2,…, max_step) at which the automata generated the desired output sequence**.
- If more than one rule generated the desired output sequence **on or before the maximum step number**, list a pair for each rule number that generated the desired output, in order, from lowest automata number to highest automata number with a space between each output pair.
- If none of the automata generate the particular line from the input file **before or on the maximum step number**, output **LINE # NONE** (where # represents the number of the input line with the search string).

Sample Input

```
3 WBWBWBWBW
1000 WBWBWBWBBBW
5235 WBWWBWWBBBBWWBBWWWWWWBBWWWBBWWWWWWBBWWBBBBWWBWWBW
5 WBWBDCWBW
END OF INPUT
```

Sample Output

```
LINE 1
(8,2)(32,2)(40,2)(64,2)(72,2)(96,2)(104,2)(128,2)(136,2)(160,2)(168,2)(192,2)(200,2)(224,2)(232,2)
LINE 2 (15,8)(158,11)(159,14)(243,8)
LINE 3 (129,84)(161,84)
LINE 4 NONE
```

Jill has been investing in a mutual fund for a while. Since her income has varied, the amount of money she has added to the investment has varied, and she hasn't always added to the investment at regular intervals. Nevertheless, she does have a complete record of the amounts she has invested, and the dates of those investments.

Periodically Jill gets a report that indicates the total value of her investment. She wonders if she would have done better by investing her money in a savings account that pays a fixed interest rate. But to determine the answer to this question, she needs to know what the equivalent interest rate would have been paid on the mutual fund, had it paid a fixed rate. You are going to help her.

For simplicity we will assume that Jill added money to her mutual fund only at the beginning of a month, and that all months have the same length. We will further assume that the interest she would have been paid had she invested in a savings account would have been paid at the end of the month, and would have been compounded monthly.

Let's consider a simple example. Suppose Jill invested \$100 at the beginning of January and another \$100 in March. At the end of April she finds that the value of her mutual fund is \$210. If the equivalent fixed monthly interest rate was *i*, then we know that at the end of January the value would have been $100 \times (1 + i)$. At the end of February the value would have been $100 \times (1 + i) \times (1 + i)$, or $100 \times (1 + i)^2$. At the end of March, the value would have been $100 \times (1 + i)^3 + 100 \times (1 + i)$, and at the end of April, the value would have been $100 \times (1 + i)^4 + 100 \times (1 + i)^2$. So the question to be answered in this case is this: what is the value of *i* such that $100 \times (1 + i)^4 + 100 \times (1 + i)^2 = 210$? The answer for this case is close to `0.016351795234`.

The Input (from file i.in)

The input from file i.in will contain multiple cases. The input for each case will begin with an integer *N* (no larger than 12) that indicates the number of times Jill invested in her mutual fund. This will be followed by *N* + 1 pairs, each pair containing an integer and a real number. The integer represents a month number (1 or larger) and the real number represents a dollar amount. The first *N* pairs give the month and amount of each of Jill's *N* investments in the mutual fund, and the last pair indicates the value of the investment at the end of the specified month. There will be one or more whitespace characters (blanks, tabs, and/or ends of lines) between the input numbers. You may assume that the month numbers are given in ascending order.

Input for the last case will be followed by a single integer –1.

The Output (to stdout)

For each case, display the case number (they start with 1 and increase sequentially) and the equivalent fixed monthly interest rate Jill's mutual fund would have paid. Display this number with five fractional digits, rounded to the nearest decimal place. You may assume the interest rate will be no less than 0 and no larger than 1. Separate the output for consecutive cases by a blank line.

Sample Input

```
2   1   100.00    3
100.00    4   210.00

3
1 100.00
2 50.00
5 200.00
7 358.41

  -1
```

Sample Output

```
Case 1: 0.01635

Case 2: 0.00520
```

Problem J: A Card Trick

The following card trick is performed by a Magician and her Assistant. The Assistant asks a member of the audience to choose 5 cards from a standard deck of 52 cards (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K of C[lubs], D[iamonds], H[earts] and S[pades]). The Assistant returns one of the cards to the audience member and then hands the remaining cards to the Magician, one at a time. After suitable mumbo-jumbo, the Magician identifies the (fifth) card held by the audience member.

The Magician determines the card as follows:
The order of the cards in the deck is determined first by the value and for cards of the same value by the suit (both in the order given above). So the total order of cards is: AC, AD, AH, AS, 2D, … , KH, KS

1. Remember the suit and value of the first card.
2. Among the remaining three cards find the position of the smallest card (in the above order). Add this position (1, 2, or 3) to the value of the first card.
3. If the larger two of the last three cards are not in order, add 3 to the result of step 2.
4. The missing card has the same suit as the first card and value that computed in step 3 wrapping around if necessary.

For example:
QH, 10D, 10C, 4D
Smallest of the last 3 cards is 4D in place 3. 10D and 10C are out of order so add 3 + 3 to Q. Wrapping around the missing card is 5H.

This problem is to write a program to perform the function of the Assistant.

The Input (from file j.in)

The first line of the input file named j.in consists of a positive integer ***n***, which is the number of datasets that follow. Each of the ***n*** following lines contain one data set. The dataset is a sequence of 5 cards separated by a space. Each card is given by a one or two character value and a one character suit as described in the first paragraph.

The Output (to stdout)

For each dataset, the output on separate lines is an ordering of the 5 input cards as shown in the sample output. The first card in the sequence is the card to be returned to the audience member. The remaining cards are those given to the Magician (in the order given to the Magician). There may be more than one solution to some problems (but that is not to say there *will* be). In cases such as these, any of the correct solutions will be accepted by the judges. For instance 10D 4D QH 10C 5H is also a solution to the first problem below.

Sample Input
```
2
4D 5H 10C 10D QH
7H 5C KS 6C 8D
```

Sample Output
```
Problem 1: 5H QH 10D 10C 4D
Problem 2: 6C 5C 7H 8D KS
```