Computer Science Competition
District 2024
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Ada
Problem 2	Ariel
Problem 3	Bodhi
Problem 4	Caroline
Problem 5	Christie
Problem 6	Claudius
Problem 7	Garold
Problem 8	Hannah
Problem 9	Jennifer
Problem 10	Leah
Problem 11	Lucas
Problem 12	Veda

1. Ada

Program Name: Ada.java **Input File: None**

Ada Lovelace, a.k.a Augusta Ada King and the Countess of Lovelace and the daughter of one of the greatest British poets Lord Byron, was also a pioneer in the discipling of computing. She worked with Charles Babbage in the 1830s on his Analytical Engine which was a follow-on to his simpler Difference Engine, a mechanical calculator.

Ada recognized that the Analytical Engine could be "programmed", making it a general-purpose computer. Even though she worked with a mechanical device, not electronic as are modern computers, and her "programming" involved mechanical switches, she is recognized as the world's first programmer!

Write a program that displays the following exact message at the left edge of the screen:

`Ada Lovelace – World's First Computer Programmer!`

Input: None.

Output: Exact statement shown above.

Sample input: None

Sample output:

```
Ada Lovelace – World's First Computer Programmer!
```

2. Ariel

Program Name: Ariel.java **Input File: ariel.dat**

Your friend Ariel is an architectural student studying subway systems, and they need your help with their homework. They need to plan out a subway stop, and they have been given the times that all the trains will be arriving and departing from the stop. Write a program to determine the minimum number of train stops required to ensure that there are no delays, in other words, every train should arrive when there is at least one open stop.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of two lines of space separated strings denoting the arrival times of all trains on the first line, and departure times of all trains on the second line, all in the format `"H:MM"`, and all minutes will be multiples of 5. The i^{th} index in both lists correspond, as in, all arrival times and departure times in the same index in their respective lists refer to the arrival and departure of the same train. There will never be two trains with the same arrival AND departure time, although trains may share the same arrival OR departure. It can be assumed that the trains operate on a 24-hour cycle, so trains arrive and leave at the same time every day.

Output: Output the integer denoting the minimum number of train platforms required so that the current train schedule will have no delays for trains when arriving. If one train arrives at the same time that another train departs, then you will only need one platform (the train engineers have been specifically trained for these situations at the same school as those two guys from the Polar Express).

Sample input:

```
3
9:30 9:45 9:50 10:30 11:30 12:00
10:00 10:05 10:15 11:00 12:00 12:10
0:00 1:00 2:00 3:00 4:00
0:10 1:10 2:10 3:10 4:10
8:15 8:25 8:30 8:35 8:40 8:45 9:00
8:25 8:40 8:45 8:45 8:55 9:00 9:30
```

Sample output:

```
3
1
3
```

3. Bodhi

Program Name: Bodhi.java **Input File: bodhi.dat**

Bodhi's older sister is studying finance in college and was showing Bodhi the concept of financial compounding. It is a type of investment, like a savings account, where the profit earned is put right back into that same investment. The result is you earn even more profit from the previous profit in addition to profit from the original investment. He thought that sounded rather interesting but wants to see the concept in action, meaning, **show me the money!**

With **PV** as the present value or a fixed amount that is invested only one time and **FV** as the future value after **n** compounding periods (addressed below), the simple compounding formula is:

$$FV = PV (1 + rate)^{n}$$

The compounding period could be days, months, quarters, years, or some other fixed period of time and defines when interest profit is calculated and put back into the account. It can get confusing comparing options so most investments state an annual percentage rate (APR) which is the result of the periodic compounding after 1-year.

For the formula above, **rate** is the APR divided by the number of compounding periods in a year. It is a periodic rate that matches the compounding period. To obtain a monthly rate, simply divide the APR by 12 monthly periods in a year and a quarterly or 3-month rate would be APR divided by 4. In addition, the standard formula requires the percentage rates be converted into their equivalent decimal form so a 5.25% rate becomes 0.0525.

The total profit after **n** periods would simply be the difference between the future value (**FV**) which is the end of the investment and the original investment (**PV**) which is the initial investment.

Input: First line will contain an integer T with $1 \leq T \leq 10$, the number of test cases. Each test case will consist of one set of whitespace-separated investment parameters on a single line. The investment parameters are **PV**, a dollar and cents amount no larger than $1,000,000 followed by an APR which is a percentage greater than 0.00% and will not exceed 25.00% (which would be a dream rate!). The final pieces of data for a single test case are the number of periods in a year in the range [1, 366] and **n**, the number of periods to compound which will not exceed 100.

Output: Each test case will produce 1 line of output containing the computed **FV** which is a dollar and cents amount and the total profit, neither of which will not exceed $3,000,000.00. Format both values with a leading dollar sign ($) and round to 2 decimal places of accuracy and separated by a single space as shown in the sample output.

Sample input:

```
3
3500.00 5.25 12 15
100.00 7.95 4 40
9999.99 9.99 2 20
```

Sample output:

```
$3736.86 $236.86
$219.72 $119.72
$26507.69 $16507.70
```

4. Caroline

Program Name: Caroline.java **Input File: caroline.dat**

Caroline's teacher in her AP Psychology class told her that if someone is asked to select a random number, it is more likely that the number will be an even number. Her class ran an experiment to verify this, and they discovered it was true.

She got to thinking. What if ten people selected a random number, would the sum of the even numbers be greater than the sum of the odds?

So, she has asked you to write a program to answer that question. Your job is to read in a list of ten whole numbers. Find the sum of the odd numbers in the list. Also find the sum of the even numbers in the list. Compare the two sums and let the world know your findings.

Input: Line #1 will consist of one integer N in the range [1,25] which indicates how many lines of data will follow. Each of the N lines of data will contain ten whole numbers in the range [0,9999]. The numbers in each data set will be separated by one whitespace.

Output: Output one of the three messages for each date set.

- If the even sum is greater, print "`Evens win by ? point(s)`" where ? is filled with the positive difference between the two sums.
- If the odd sum is greater, print "`Odds win by ? point(s)`" where ? is filled with the positive difference between the two sums.
- If the sums are equal, print the message "`It's a tie!!!`"

Sample input:

```
5
12 13 14 15 23 24 25 26 55 62
7 7 7 7 7 7 7 7 28 28
2 3 2 3 2 3 2 3 2 3
5 1 2 8 6 7 5 3 0 9
11 22 33 44 55 66 77 88 99 110
```

Sample output:

```
Evens win by 7 point(s)
It's a tie!!!
Odds win by 5 point(s)
Odds win by 14 point(s)
Evens win by 55 point(s)
```

5. Christie

Program Name: Christie.java **Input File: christie.dat**

Christie is so intrigued with numbers and their properties. She of course adores prime numbers (and who doesn't?), but she also is very interested in other types of numbers like Fibonacci Numbers, Happy Numbers, Evil Numbers, Fermat numbers, etc. She is determined to create her own special numbers and name them Christie Numbers.

So, this is what she has decided to do.

A Christie Number is a whole number in which the sum of the squares of the digits is a perfect square.

Example: 148 is a Christie Number:

$$1^2 + 4^2 + 8^2 = 1 + 16 + 64 = 81$$

Since 81 is a perfect square, 148 is a Christie Number.

Christie will give you two integers A and B where you are guaranteed that A < B. Your task is to check every natural number from A to B (inclusive) and to list the Christie numbers in that range in order from smallest to greatest.

Input: Line #1 will consist of one integer N in the range [1,25] which indicates how many lines of data will follow. Each of the N lines of data will contain two integers A, then B. Both numbers are in the range [1,9999].

Output: Output a list of Christie Numbers in the range [A,B] written horizontally with one white space in between each. If there are no Christie numbers in that range, print "NONE".

Sample input:

```
5
10 20
30 50
100 200
50 55
41 42
```

Sample output:

```
10 20
30 34 40 43 50
100 122 148 184 200
50
NONE
```

6. Claudius

Program Name: Claudius.java **Input File: claudius.dat**

You and Claudius are lost in the woods! Quick, write a program on your handy dandy pocket computer to find the shortest path out of the woods! You have a map of the woods, with different geological features and the parking lot marked, and you need to determine if you will escape from the woods before the Forest Rangers come to find you. This map is magic, and will also contain the locations of certain dangerous animals and areas to avoid. The map will be made up of the following characters:

- `'M'` – denotes the location of a mountainous region on the map, these areas can be crossed at a rate of 3 hours per space.
- `'T'` – denotes the location of a forested area of the map, these areas can be crossed at a rate of 2 hours per space.
- `'R'` – denotes the location of a rock/boulder, these areas are impassable.
- `'Q'` – denotes the location of quicksand, these areas are impassable, unless they are directly adjacent (up, down, left, right) to a forested area, in which case you can cross at a rate of 3 hours per space, as you can use the trees to climb out if you get stuck.
- `'V'` – denotes the location of a river, these areas are impassable.
- `'A'` – denotes the location of an alligator, which you must stay at least one block away from (you cannot be adjacent in any direction including diagonals).
- `'.'` – denotes a path/trail/dirt patch which can be passed at a rate of 1 space per hour.
- `'S'` – denotes your starting point on the map.
- `'E'` – denotes the parking lot, which is the end point of your journey.
- `'B'` – denotes the location of a bear, which you must be at least 2 spaces away from at all times, including diagonals.

You can only move in the 4 cardinal directions (up, down, left, right).

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will begin with 3 space-separated integers, `r`, `c`, and `h`, denoting the number of rows and columns in the map of the woods, and the number of hours you have until the Forest Rangers come looking for you. The following `r` lines will each contain `c` characters denoting the map of the woods.

Output: If you make it to the parking lot in `h` or less hours, output the string `"Free at last, Free at last. "`, followed by the number of hours you had left until the Forest Rangers will look for you (could be 0), followed by the string `" hour(s) to spare."`. If you do not make it to the parking lot in time, output the string `"Smokey the Bear is en route."`.

Sample input:

```
2
6 7 15
S..MM.V
..MMMMV
...MMRV
VV.VVVV
A...QQE
MMM....
5 5 12
SMTTB
MMTTT
M..TT
VVVVV
ARR.E
```

Sample output:

```
Free at last, Free at last. 3 hour(s) to spare.
Smokey the Bear is en route.
```

7. Garold

Program Name: Garold.java **Input File: garold.dat**

You and your friend Garold have been playing a game called super tic tac toe. The main issue you have been having is that the board is somewhat confusing and you never know who's won. You need to write a program to take in a given super tic tac toe board and find out who has won. The rules of super tic tac toe are as follows:
Each super tic tac toe board will be made up of 9 regular tic tac toe boards, arranged with one regular board making up a cell. In order to win super tic tac toe, you need to win 3 of the regular boards in a row (so if you win the top 3 boards you win super tic tac toe). You can win if you get all of any column, row or either of the diagonals. To win each board, you need to win a row, column or diagonal, just like in regular tic tac toe. You need to output who wins the game.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of 9 lines of 9 characters each, made up of `'X'` – denoting an X on the board, `'O'` – denoting an O on the board, `'.'` – denoting an empty space on the board. Every 3x3 characters of input denote a board (top left 3x3 denotes the top left board on the super tic tac toe board, and so on). Each 3x3 board will have a maximum of one winner, but there is no guarantee that the overall board is won by either team.

Output: If one of the players has won the game, output the string `"Player "`, followed by an `X` or an `O`, denoting which player won, followed by `"Won."`. If neither player has won the game, output `"Cat's Game."`. After outputting this string, output the the layout of the super board, with each 3x3 board denoted by an `X`, `O`, or `.`, denoting the winner if the board was won, or a `.` if no one has won the board. See the Sample output for more information.

Sample input:

```
2
X.OOOOXO.
OX..X.XX.
OOXX..OOX
X.O..X.OO
XOO.O.OO.
X..OO.OXX
XXXOOO.OX
OO..X.XO.
O.XXOXOOX
X.OOOOXO.
O...X.XX.
OOXX..OOX
X.O..X.OO
.OO.O.OO.
X..OO.OXX
X.OOO..OX
OO..X.XO.
O.XXOXXXX
```

Sample output:

```
Player X Won.
XOX
X.O
XOO
Cat's Game.
.OX
..O
O.X
```

8. Hannah

Program Name: Hannah.java **Input File: hannah.dat**

Hannah is practicing her programming skills for the upcoming UIL Computer Science contest season. She heard that the State contest experience is very challenging and wants to be prepared. Her coach explained how the overall state scores are grouped by classifications (1A, 2A, 3A, 4A, 5A, 6A) and are a combination of both programming and the written test. Each contest team consists of 3 students that take the written test with 40 questions which has a max score of 240 but could be negative when students answer too many questions incorrectly; there is no penalty for unanswered questions. The max team score for the written test is 720, or 3 scores of 240 points. The programming component of the contest consists of 12 problems worth a max of 60 points each but submissions that are not correct reduces the problem's max score by 5 points for each bad submission. The team programming score is another 720 points, or 12 programs of 60 points. The overall team score is simply the sum of the written and programming scores.

The following table is a sample of the programming data.

Prog Scores	Class	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Prob 6	Prob 7	Prob 8	Prob 9	Prob 10	Prob 11	Prob 12
Team 1	2A	55	60	50	60	60	60	60	60	60	60	0	60
Team 2	5A	60	60	60	60	60	55	60	60	50	0	60	55
Team 3	6A	60	50	60	55	60	60	60	60	0	60	60	45

The following table is a sample of the written test data.

Test Scores	Student 1	Student 2	Student 3
Team 1	200	210	184
Team 2	104	172	224
Team 3	86	164	196

Hannah has accepted a challenge of processing the raw scores to determine the top 3 teams in each classification. It is just summing the scores and finding the 3 top overall team scores in each classification. Ties are not common so we will ignore that possibility for this program. Can you handle her challenge?

Input: First line will contain an integer **T** with $1 \leq \mathbf{T} \leq 10$, the number of test cases. Each test case will start with a positive integer **N** which is the total number of teams. The **N** following lines will each contain a school name with no spaces, a classification as shown above, and 12 integers in the range [0,60], all items whitespace-separated. Those lines are then followed by **N** more lines each containing a school name with no spaces and 3 integers in the range [−80,240], all whitespace -separated. Team names will be in the same order for both sets of scores but classification levels may vary in order. There is guaranteed to be 1 or more teams for each classification and they must have 12 program scores and 3 written exam scores.

Output: Each test case will produce a list of the team names and scores in descending order for each classification, organized from 1A to 6A. Label and format the results as shown in the sample below. Display a single line following each test case containing 15 equal signs "==========".

Sample input:

```
2
15
Team_1 2A     60     40     30     60     55     50     30     30     55     35     50     0
Team_2 5A     35     50     30     45     50     50     30     60     50     50     0      30
Team_3 6A     35     30     45     45     30     35     30     40     30     0      40     50
Team_4 1A     30     40     0      55     0      30     35     55     40     55     30     60
Team_5 5A     40     55     30     45     55     45     40     0      35     60     40     55
```

~ Input data continues on next page ~

~ Hannah input continued ~

```
Team_6 4A       0       35      45      40      30      30      0       50      45      55      40      50
Team_7 6A       35      0       45      55      60      0       60      50      55      30      30      50
Team_8 1A       35      45      35      60      40      35      55      40      0       30      45      55
Team_9 6A       30      35      55      0       40      60      30      50      60      35      35      55
Team_10         4A      35      45      0       60      0       40      30      45      55      35      35      60
Team_11         3A      45      0       35      35      45      0       30      55      40      40      60      30
Team_12         5A      0       30      55      45      60      55      0       30      60      35      60      55
Team_13         3A      60      0       55      50      35      30      40      0       50      45      35      35
Team_14         5A      50      45      0       60      40      40      45      40      0       60      45      45
Team_15         2A      35      55      60      0       60      50      60      35      30      0       45      35
Team_1 184      155     70
Team_2 199      192     203
Team_3 136      177     229
Team_4 93       75      121
Team_5 0        210     228
Team_6 180      220     131
Team_7 174      92      235
Team_8 226      55      234
Team_9 92       196     163
Team_10         234     170     145
Team_11         136     178     185
Team_12         185     112     84
Team_13         68      -12     116
Team_14         230     121     146
Team_15         75      97      53
20
Team_1 2A       50      30      55      30      35      55      30      40      55      35      30      0
Team_2 5A       60      30      35      60      45      50      30      30      30      35      0       55
Team_3 6A       50      60      50      50      45      55      45      50      35      0       55      55
Team_4 1A       30      30      40      30      30      30      50      35      0       40      35      60
Team_5 5A       30      50      35      55      30      30      40      0       50      45      40      50
Team_6 4A       0       35      30      40      55      55      0       35      60      60      60      50
Team_7 6A       45      0       35      30      35      0       60      55      30      30      60      60
Team_8 1A       45      60      0       40      0       45      35      30      60      30      60      60
Team_9 2A       30      35      30      0       35      30      30      55      45      45      30      60
Team_10         4A      40      50      0       60      0       55      35      35      40      60      60      60
Team_11         3A      35      0       45      30      35      0       45      35      35      30      45      45
Team_12         5A      0       45      45      45      50      45      0       35      40      45      55      40
Team_13         3A      60      0       50      30      55      50      50      0       30      35      35      55
Team_14         5A      40      55      0       40      35      30      30      50      0       35      50      45
Team_15         2A      40      30      50      0       30      50      60      30      55      0       50      30
Team_16         4A      60      55      55      60      0       50      50      30      30      35      0       55
Team_17         3A      30      55      50      55      50      0       40      40      55      55      50      0
Team_18         6A      40      60      30      50      35      45      0       30      40      55      0       60
Team_19         5A      35      30      50      45      55      40      40      0       45      0       40      45
Team_20         6A      40      30      40      40      50      45      35      55      0       55      30      60
Team_1 98       222     156
Team_2 134      208     140
Team_3 215      128     62
Team_4 180      67      132
Team_5 0        206     210
Team_6 148      141     75
Team_7 218      202     174
Team_8 73       205     231
Team_9 112      103     187
Team_10         193     226     131
Team_11         59      138     193
Team_12         94      176     205
Team_13         210     -12     75
Team_14         231     133     200
Team_15         237     183     121
```

~ Input data continues on next page ~

~ Hannah input continued ~

```
Team_16      -2     113    196
Team_17      144    99     133
Team_18      168    56     226
Team_19      64     73     57
Team_20      195    93     235
```

Sample output:

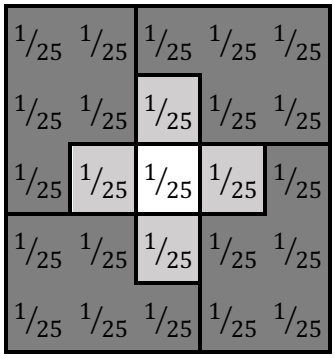
```
Classification 1A Results
Team_8 990
Team_4 719
Classification 2A Results
Team_1 904
Team_15 690
Classification 3A Results
Team_11 914
Team_13 607
Classification 4A Results
Team_10 989
Team_6 951
Classification 5A Results
Team_2 1074
Team_14 967
Team_5 938
Team_12 866
Classification 6A Results
Team_7 971
Team_3 952
Team_9 936
========================
Classification 1A Results
Team_8 974
Team_4 789
Classification 2A Results
Team_15 966
Team_1 921
Team_9 827
Classification 3A Results
Team_17 856
Team_11 770
Team_13 723
Classification 4A Results
Team_10 1045
Team_6 844
Team_16 787
Classification 5A Results
Team_14 974
Team_2 942
Team_12 920
Team_5 871
Team_19 619
Classification 6A Results
Team_7 1034
Team_20 1003
Team_3 955
Team_18 895
========================
```

9. Jennifer

Program Name: Jennifer.java **Input File: jennifer.dat**

While studying infinite sums in her Calculus class, Jennifer was introduced to the well-known proof which shows that $0.99\overline{9} = 1$. Being well-versed in the world of mathematical bases, Jennifer's Calculus teacher showed them a similar proof to show what this would be equivalent to in a base five numbering system.

The idea is that if you take a square with side length of 1, and break it down into 25 congruent squares, you can form four identical sections, each with 5 of the congruent squares. This would mean that each of those sections comprise 1/5th of the area of the whole square. Doing so would also leave you with 5 remaining squares. If you were to take 4 of the remaining 5 squares, you would then have four new sections each comprising 1/25th of the area of the whole square. Then, take the last remaining square, and perform the entire process again on that square infinitely.



This means that $4\left(\frac{1}{5}\right) + 4\left(\frac{1}{25}\right) + \cdots + 4\left(\frac{1}{5}\right)^n = 1$. In other words, for a base five numbering system, $(0.44\overline{4})_5 = 1$. However, Jennifer was interested to see whether or not this property of infinite sums generalized to different bases. While investigating this, Jennifer discovered a generalized formula which showed that for all $|x| \geq 1$, where x is equivalent to the base, that…

$$(x-1)\sum_{n=1}^{\infty}\left(\frac{1}{x}\right)^n = 1$$

Discovering this generalized formula got Jennifer interested in the notion of non-integer bases. Specifically, given the inverse of an arbitrary base, help Jennifer determine the most simplified number of sections that will be required at each stage in the infinite sum process.

Input: The first line will be a single integer T ($1 \leq T \leq 100$) denoting the number of test cases to follow. The next T lines will consist of 2 space-seperated integers, n and d ($1 \leq n, d \leq 2^{31} - 1$), denoting the numerator and the denominator of the inverse of the current base. It is guaranteed that the value of $\left|{}^{d}/_{n}\right| > 1$.

Output: For each of the T test cases, output two space-separated integers, n_s and d_s, denoting the simplified numerator and denominator of the number of sections that are required at each stage in the infinite sum process.

Sample input:

```
2
3 4
341 1054
```

Sample output:

```
1 3
23 11
```

10. Leah

Program Name: Leah.java **Input File: leah.dat**

Leah is rather fond of expressing numbers in different bases. In particular, Leah has a real affinity for binary numbers. As such, she already has a good understanding of what binary numbers are, and how to read them. In most cases, when generating binary numbers, binary numbers are ordered from least to greatest. For example, the following is a list of the binary representation of the numbers 0 through 7 which are expressed to 3 bits:

0b000,	0b001,	0b010,	0b011,	0b100,	0b101,	0b110,	0b111
↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7

Leah in her Digital Logic class was recently introduced to Gray Codes, which are an alternative way to order binary numbers. Rather than simply adding one to the previous binary number to generate the next binary number, Gray Codes order binary numbers according to the simple principle that no two adjacent numbers can differ by more than a single bit. The following is the order of the first 8 Gray Codes expressed to 3 bits:

0b000,	0b001,	0b011,	0b010,	0b110,	0b111,	0b101,	0b100
↓	↓	↓	↓	↓	↓	↓	↓
0	1	3	2	6	7	5	4

However, generating Gray Codes can be decently difficult to do so by hand. Help Leah by writing her a program that generates Gray Codes for different bit widths.

Input: The first line of input will consist of a single integer n ($1 \leq n \leq 32$) denoting the number of testcases to follow. The next n lines will each contain a single integer w_i ($1 \leq w_i \leq 8$) denoting the width of any given binary number that Leah wants to generate.

Output: For each of Leah's n requests, on their own line, print a space-separated list of the decimal representation of the numbers 0 through $2^{w_i} - 1$ in their Gray Codes ordering.

Sample input:

```
4
3
1
4
2
```

Sample output:

```
0 1 3 2 6 7 5 4
0 1
0 1 3 2 6 7 5 4 12 13 15 14 10 11 9 8
0 1 3 2
```

11. Lucas

Program Name: Lucas.java **Input File: lucas.dat**

Lucas is a track coach keeping "track" of his runners' times in the "Uphill Mountain Running/Climbing Challenge" - the UMRCC. It is important for him to be very aware of the progress of each member of the team.

Lucas will give you a list of times in seconds for each of his runners. Your job is to examine the list and send him back the average time for each runner written in minutes and seconds.

If the seconds come out to be a decimal number, round down to the whole second. Or as he told us, "chop off the decimal."

Now Lucas realizes that everybody has a bad day and a super-great day from time-to-time. He would like you to drop the fastest and the slowest time for each runner if they have at least three times listed. If a runner has only one or two recorded times, do not drop any scores - because you can't. Each runner will have at least one time, guaranteed.

Input: Line #1 will consist of one integer N in the range [1,25] which indicates how many lines of data will follow. Each of the N lines of data will contain a list of whole number times separated by one white space. On each line there will be T numbers where T is in the range [1,20].

Output: Output the average time. Drop the fastest and slowest times if there are at least three times in the list. The time should be written in the following format. Minutes:Seconds where minutes will be an integer in the range [0,167] and seconds will be a two-digit number in the range [0,59]. There will be a colon in between the two numbers. There should be no whitespace in your answers. If seconds calculate to be a decimal, truncate the value. For example, 34.97 seconds would truncate to 34 with no decimal.

Sample input:

```
5
965 1200 1315 950
1408
2201 1534
1232 1236 1238 1240 1300 1303 1220 1251 1332 1299
1600 1200 1300 1400 1500
```

Sample output:

```
18:02
23:28
31:07
21:02
23:20
```

12. Veda

Program Name: Veda.java **Input File: veda.dat**

Veda is a successful entrepreneur and has made a name for herself by owning a company which specializes in creating custom-made signs. Like most custom-made sign businesses, Veda's company, Ingenious Insignia™, prices her signs based off of the amount of ink, dye, or vinyl that would be required to create said sign. As a result, she generates her prices on a per-letter basis, where each letter occurring on the sign costs a certain amount. However, being the smart businesswoman that she is, Veda knows that certain letters require more materials compared to others. As a result, Veda's pricing is different depending on the letter in question.

Wanting to be able to ensure that Veda maintains a competitive advantage over her competitors, she wants to figure out whether or not a certain letter-to-price ordering would make her business favorable. Help Veda write a program that, given a list of letter-to-price pairings, as well as different slogans for signs from potential future customers, returns the cost for each test sign.

Input: The first line of input will consist of a single integer n ($1 \leq n \leq 26$) denoting the number of unique letter-to-price groups. The next n lines will each consist of a comma-separated list of letters, followed by a colon, followed by a price p ($0 < p \leq 10$). This denotes that each letter contained in the comma-separated list costs p dollars per letter. Valid letters consist of only the standard 26 capitalized English letters. All punctuation, spaces, and other special characters are considered free. It is also guaranteed that each letter appears exactly once among the n comma-separated lists.

The next line will consist of a single integer m ($1 \leq m \leq 50$) denoting the number of slogans that Veda wishes to calculate the price for. The next m lines will each consist of a single slogan, which will strictly consist of capitalized English letters, whitespace, and special characters.

Output: For each slogan that Veda wishes to test, in the order that it appears in the input, on its own line, print out the cost *P* that it would take to print said slogan using the letter-to-price groupings described in the input. This dollar amount should be prepended with a dollar sign ('$') and should be expressed to two decimal values (rounded to the nearest cent).

Sample input: (*indented lines are continuation of previous line*):

```
6
B,C,D,F,G:0.023
H,J,K,L,M,N:0.102
P,Q,R,S,T:0.0098
V,W,X,Z:0.721
A,E,I,O,U:0.3400005
Y:1.23
5
UNIVERSITY INTERSCHOLASTIC LEAGUE
COMPUTER SCIENCE IS THE BEST SCIENCE
A RUBBER DUCK IS A PROGRAMMER'S BEST FRIEND!!!!
WOULD YOU RATHER HAVE UNLIMITED BACON, BUT NO GAMES, OR GAMES, UNLIMITED GAMES, BUT NO
   GAMES?
ROAD WORK AHEAD... UH, YEAH, I SURE HOPE IT DOES
```

Sample output:

```
$7.03
$4.71
$4.78
$15.21
$8.38
```