University Interscholastic League

Computer Science Competition

2014 District 2 Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

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. Incorrect submissions may be reworked and resubmitted, but will receive a deduction of 5 points for each incorrect submission. Deductions are only included in the team score for problems that are ultimately solved correctly.

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.

II. Table of Contents

1. Cows

Program Name: Cows.java Input File: cows.dat

Farmer John owns a large pasture with K cows inside. Farmer John's cows are not like most cows. They moo at regular intervals. He was interested in studying these cows, so he hired a scientist to try to identify how many cows he could expect to moo in a certain time interval. The scientist was able to establish, for each cow, the time at which they first moo, and then the length of the delay between successive moos. However, the scientist didn't get much further than that, and went insane, mooing along with the cows. Farmer John got rid of the old scientist and replaced him with you. Since you don't want to go insane, you decide to write a program that, given the information the old scientist obtained, solves the problem for Farmer John.

Input
The first line of the input will be a single integer K, the number of cows in Farmer John's field.
The next K lines of input each consist of two integers: S and I. S is the time of the first moo the cow makes, and I is the time between moos after the first one for that cow.
The next line is a single integer N, the number of intervals Farmer John is interested in.
The next N lines of input each consist of two integers: A and B, or the start and end time of each interval (inclusive).

Constraints

```
1 <= K <= 20
0 <= S <= 10000
1 <= I <= 10000
1 <= N <= 20
0 <= A, B <= 10000
```

Output
For each interval, output the number of moos that occur in that interval on its own line.

Example Input File
```
2
0 4
2 5
1
4 8
```

Example Output to Screen
```
3
```

Explanation of test case:
Each cow will moo at time t = S + k * I, where k is a non-negative integer. Or in other words, cow 1 will moo at 0, 4, 8, 12, 16, … and cow 2 will moo at 2, 7, 12, 17, ….
So for the interval 4 – 8 inclusive, cow one moos twice (at 4 and 8), and cow 2 moos once (at 7), so the total number of moos in this interval is 3.

2. Destination

Program Name: Destination.java Input File: destination.dat

You are helping blindfolded people reach their destination by telling them if they pass over their destination. You will be told the person's initial location and final location, and then be given a list of their movements. State how many moves it took for them to pass over or land at their destination or "You didn't make it" (including the quotation marks) if they never make it to their destination. Movements will take the form of "(Direction) (Distance)" where direction is north, south, east, or west. If a person starts on their destination, the answer is 0. North is +Y, South: -Y, East: +X, and West: -X.

Input

The first line will contain 't' the number of test cases to follow. The first line will contain 5 numbers "X1 Y1 X2 Y2 N" where (X1, Y1) is the starting location and (X2, Y2) is the destination location, and N is the number of directions to follow with each direction on its own line. All numbers will be integers.

Output

How many moves it took to pass over the destination or "You didn't make it"

Constraints

```
1 <= t < 10
-100 <= X1,Y1,X2,Y2 <= 100
Direction : {north, east, south, west}
0 < Distance < 30
0 < N < 30
```

Example Input File

```
2
0 0 2 4 3
west 3
north 4
east 9
1 2 4 6 2
south 3
east 1
```

Example Output to Screen

```
3
You didn't make it
```

Explanation

On the first test case the person traveled from (0,0) to (-3,0) to (-3,4) to (6,4). On the third move, the person passes over (2,4) when passing from (-3,4) to (6,4).

On the second case the person moves from (1,2) to (1,-1) to (2,-1) but never passes over (4,6).

3. Distance

Program Name: Distance.java Input File: distance.dat

You will calculate the average distance between two randomly chosen points. You will be given a number to seed a random number generator with and the number of simulations to run. Each simulation will involve generating the integers X1, Y1, X2, and Y2 in that order as values greater than or equal to 0 and less than 1000. Then calculate the distance between them, truncating that distance to 3 decimal places. For each test case, print the average distance between the two points.

Input

The first line will contain 't' the number of test cases to follow. Each test case will consist of a single line containing r and n. r is the number to seed the Random object with and n is the number of simulations to run.

Output

For each test case print out the average distance, truncated to three decimal spaces.

Constraints

```
1 <= t < 10
0 < r, n < 10^8
```

Example Input File

```
2
12087 400
7418 978
```

Example Output to Screen

```
553.994
525.789
```

4. Encryption

Program Name: Encryption.java Input File: encryption.dat

As part of an encryption system, you will be given a string of numbers using which you will then create an encryption number. This number will be taken by finding the least common multiple of the numbers.

Input

The first line will contain a number 't' less than 100 specifying the number of lines to follow. The rest of the lines will contain a list of numbers of unspecified length.

Output

For each test case print out the LCM.

Constraints

```
1 <= t < 100
0 < numbers
1 <= answer < 10^12
```

Example Input File

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

Example Output to Screen

```
36
105
```

5. Lottery

Program Name: Lottery.java Input File: lottery.dat

You bought a lottery ticket that has a long integer (the ticket number) printed on it. The drawing of the winning number is being shown on TV. To decide the winning number, the announcer pulls a random digit out of a bin, which becomes the least significant digit of the winning number. He then continues to pull digits out, and appends them to the left of the existing number, building up the suspense.

You would love to win a prize. The deal is that for every digit in your ticket's number that matches the digit in the same position (counting from the rightmost position) of the winning number, you win $10. Note that the winning number may have more or fewer digits than your own number. The match has to be true for position and value.

Write a program to calculate your winnings.

Input
The first line has the number of testcases, `T`. `T` lines follow, one line for each testcase.
Each testcase contains two integers `N` and `M`, separated by a space. The numbers will not have leading zeros followed by a non-zero digit.

Output
For each testcase, print out your winnings in dollars.

Constraints
```
0 <= N <= 9,999,999,999
0 <= M <= 9,999,999,999
```

Example Input File
```
3
42382 342
123456 654321
380 0
```

Example Output to Screen
```
$20
$0
$10
```

6. Memory

Program Name: Memory.java Input File: memory.dat

Java only has "references," but other programming languages like C have "pointers." In this assignment you will implement these pointers. A variable has a value and an address associated with it. For example a variable '**var1**' might have an address 10 and a value of 20. Another variable '**var2**' might have an address of 20 and a value of 10. These variables can be used to hold values or point to another variable. The address, value, or pointed to value can be obtained through the prefix '&', no prefix, and '*' respectively.

Using the variables var1 and var2 above, examples are as follows:

&var1 will return var1's address (10); &var2 will return 20.

*var1 will return the value located at the address which var1 points to. *var1 returns the value at address 20 (10). *var2 returns 20.

var1 returns 20, and var2 returns 10.

Input

The first line will contain n, the number of test cases. n test cases will follow. Each test case will start with a line with two integers, m and k, separated by a space. m lines will follow in the format [var] [address] [value] where var is the variable name and the address and value are numbers. Next k lines will follow with formulas.

Output

For each formula, evaluate the answer and print out that value on a separate line, based on that test case's variables.

Constraints

```
0 <= n <= 20
1 <= M <= 100
1 <= k <= 100
```

Example Input File

```
1
3 3
Var1 20 40
Var2 40 30
Var3 30 20
&Var2
*Var1
Var3
```

Example Output to Screen

```
40
30
20
```

7. Product of Divisors

Program Name: ProdDiv.java Input File: proddiv.dat

Given two positive integers `A` and `B`, find all numbers `X` such that `A <= X <= B`, and the product of all positive divisors of `X` is prime. A divisor of `X` is a number `N` such that `X%N = 0`.

Input
The first line has the number of testcases, `T`. `T` lines follow, one line for each testcase. Each testcase line contains `A` and `B (B >= A)` separated by a space.

Output
For each testcase, print out the number of integers X that meet the above condition.

Constraints
```
1 <= T <= 10
1 <= A <= 1000
1 <= B <= 1000
```

Example Input File
```
2
2 5
7 7
```

Example Output to Screen
```
3
1
```

8. RNA

Program Name: RNA.java Input File: rna.dat

You are taking an intro biology class, and you are learning about RNA transcription. In this process, an RNA strand must match up to an equal length DNA strand, as defined by the following matching.

At each position, a 'T' in the DNA strand must match with an 'A' in the RNA strand, an 'A' in the DNA strand must match with a 'U' in the RNA strand, a 'C' in the DNA strand must match with a 'G' in the RNA strand, or a 'G' in the DNA strand must match with a 'C' in the RNA strand. If these matchings occur at every position, the pair is deemed "good". If one or more positions are not matched, or any letters besides these occur, the pair is deemed "bad".

Your homework has a number of problems, each of which is checking whether a given DNA strand matches to a given RNA strand. However, your teacher has given you some strands of length up to 200! So, rather than spend forever on your homework, you write a program to do it for you.

Input
The first line will be a single integer T (< 10), the number of problems on your homework.
The first line of each problem will be a single integer N (0<=N<=200), the length of the strings for this problem.
The following two lines will each contain a single string, each of length N. The first line will be the DNA strand, and the second line will be the RNA strand.

Output
For each problem, output "GOOD" if the strings only consist of the letters A, T, G, C, and U, and each base pair is matched properly between the strands. If one of these is not true, output "BAD" instead.

Example Input File
```
3
4
ATGC
UACG
4
ATGC
CGUA
3
AGQ
UCF
```
Example Output to Screen
```
GOOD
BAD
BAD
```

Explanation: In the first case, all 4 indexes are valid pairs, so the answer is good.
In the second case, the first pair, A-C is not valid, so the answer is bad.
In the third case, the last pair, Q-F, is not valid, so the answer is bad.

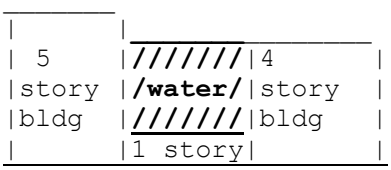
9. Skyline

Program Name: Skyline.java Input File: skyline.dat

As you sit in your class, you begin doodling a 2D city skyline on your graph paper. As it is raining outside, in your 3D world, you wonder what would happen if it rained within this 2D city – assuming that the rain came magically from the top of your paper.

As your curiosity grows, you want to determine how much water would remain if it rained for an excessively long time. The story height of each building in the skyline is represented by an integer in an array. Assume that each building is built right up against the next building, water tight, and that there is an invisible watertight barrier across the front and back of all buildings, (but not at the ends), as tall as the highest structure, that would hold the water in. Also assume that enough water has fallen is to completely fill all spaces between all buildings.

For example, if your buildings were of story height 5, 1, and 4, then there would be 3 stories of water that accumulated between the buildings.



Input

The first integer T will represent how many test cases there are to follow. For each test case, there will be a line containing a single number n, which represents the number of building spaces in your city. The next line contains n integers, which represent the block height of each building, which can range between 0 and 100.

Output

For each test case print out the number of blocks that the water would take up.

Example Input File

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

Example Output to Screen

```
3
12
0
```

10. Sticks into Polygons

Program Name: Sticks_Polygon.java **Input File: stickspolygon.dat**

You are given a box of sticks of different lengths. What is the maximum number that you can use to form a polygon? The polygon's area has to be strictly > 0.

Input
The first line has the number of testcases, `T`. 2*`T` lines follow, two lines for each testcase.
The first line of each testcase line contains `N`, the number of sticks you have. The next line contains `N` integers separated by spaces, the length of each stick.

Output
For each testcase, print out the maximum number of sticks that can be used to form a polygon of area > 0. If no polygon can be formed, print `-1`.

Constraints
```
1 <= T <= 10
3 <= N <= 20
1 <= Lengths <= 20
```

Example Input File
```
3
3
1 2 3
4
4 2 1 3
4
1 2 8 2
```

Example Output to Screen
```
-1
4
3
```

Explanation for the third testcase
A triangle can be formed from sticks of length 1, 2, and 2, excluding the stick of length 8.

11. Stock

Program Name: Stock.java Input File: stock.dat

The stock market is a complicated place. In theory, the algorithm to make money is easy: sell high and buy low. However, given that you do not know the future, buying and selling at the correct time is much more difficult. Fortunately, you found a magical oracle that tells you exactly what the stock prices of a given company will be for the next N days at closing time. Given this, determining when to buy and sell optimally (at the local minimum/maximum) is easy. Write a program that, given an N day schedule, determines when to buy and sell the stock. Assume that you only buy and sell on these fixed stock prices, we ignore the fluctuations during the day.

Additionally, for the sake of determining a local minimum/maximum, assume the stock price is never the same on consecutive days. Also, assume you already have some stock in each of these companies, so you don't have to buy before you can sell the first time.

Input
The first line will be a single integer T, the number of companies your oracle has fetched stock schedules for.
The first line of each company will contain a single integer N, the number of days of the schedule, followed by a string that is the name of the company. The name will have no spaces and consist of only capital letters.
The next line of each will contain N space-separated integers, the stock price on the ith day, for $1 \leq i \leq N$.

Output
For each company, print the company name on a single line. Then for each action, either buy or sell, print the day number it is performed, followed by the action. These should be printed in day order.

Example Input File
```
2
4 TEST
1 3 2 4
5 ABC
5 3 1 4 8
```

Example Output to Screen
```
TEST
1 BUY
2 SELL
3 BUY
4 SELL
ABC
1 SELL
3 BUY
5 SELL
```

Explanation of first test case:
At day 1, the value of the stock on day 1 (1) is less than the value of the stock on day 2, and since there is no previous day to compare it to, it is a local minimum, so we want to buy.
At day 2, the value of the stock on day 2 (3) is greater than the value of the stock on day 1 (1) and day 3 (2), so it is a local maximum, so we want to sell.
At day 3, the value of the stock on day 3 (2) is less than the value of the stock on day 2 (3) and day 4 (4), so it is a local minimum, so we want to buy.
At day 4, the value of the stock on day 4 (4) is greater than the value of the stock on day 3 (2), and there is no day 5 to compare against, so it is a local maximum, so we want to sell.

12. Texas

Program Name: Texas.java Input File: None

Print the names of the Capital of Texas, the Texas State Flower, the Pro-football team from Dallas, and our state's nickname.

Input
N/A.

Output
See the Example output.

Example Input File
```
N/A
```

Example Output to Screen
```
Austin
Bluebonnet
Dallas Cowboys
The Lone Star State
```