2017 FHSPS Playoff
February 25, 2017
Timber Creek High School

Problem	Filename	Problem Name
a	average	At Least Average
b	bonus	Bonus Points
c	counting	Counting Factors
d	dragons	Defeating Dragons
e	electric	Electric Car Race
f	frogger	Frogger for Four Year Olds
g	game	Ground Game
h	highscore	High Score
i	igloo	Igloo Construction
j	just	Just in Case (the set is finished)

At Least Average

Filename: *average*
Time Limit: *8 seconds*

Blake plays a video game that has *n* levels. On each level, she can earn in between 0 and 10 points, inclusive. The number of points she gets on a level must be an integer. She has set up a goal for herself to equal or beat a particular average. In order to make her performance seem impressive, she's decided that she wants to count the number of intervals (sets of consecutive levels) where she's equaled or beaten her target average.

We define an interval [i, j], with $1 \leq i \leq j \leq n$, to be each level starting at level i and ending at level j, including level j. For example, if Blake played 5 levels with her scores being 5, 1, 2, 4 and 6, and Blake's target average was 3, then here are the following intervals where she equaled or beat her target average:

[1, 1] with average 5
[1, 2] with average 3
[1, 4] with average 3
[1, 5] with average 3.6
[2, 5] with average 3.25
[3, 4] with average 3
[3, 5] with average 4
[4, 4] with average 4
[4, 5] with average 5
[5, 5] with average 6

Blake can make the impressive statement that on 10 intervals her average score was 3 or more.

The Problem

Given the number of levels Blake has played in her video game, her scores on each level, and the average value she'd like to equal or beat, determine the number of intervals that she equaled or beat the given average.

The Input

The first line of input will consist of a single positive integer, *v* ($v \leq 10$), representing the number of input cases to process. Input for each case follows, one case taking two lines. The first line of input for each case contains two positive integers, *n* ($n \leq 100000$), and *a* ($a \leq 10$), separated by spaces, representing the number of levels of the video game and the average Blake wants to obtain, respectively. The following line contains *n* space separated integers, each in between 0 and 10 inclusive, representing Blake's score on each of the *n* levels, in order.

The Output

For each input case, output a single integer on a line by itself representing the number of intervals where Blake obtained her desired average or higher.

Sample Input

```
2
5 3
5 1 2 4 6
9 6
10 1 10 1 10 1 10 1 10
```

Sample Output

```
10
15
```

Bonus Points

Filename: *bonus*
Time Limit: *1 second*

Geo has designed a rather interesting video game with bonus points. Her game has levels and at the end of each level, if a player's point total has exceeded a particular threshold, she receives some bonus points for that threshold. Each different game has a critical even integer value, *c*. In a game, a bonus is received for reaching *c* points, then 2*c* points, followed by 4*c* points, and so forth. In general, the k^{th} bonus level occurs when a player reaches a total of $c2^{k-1}$ total points. The bonus is always precisely half of points necessary to reach that bonus level. Thus, the k^{th} bonus is $c2^{k-2}$ points. At the end of a round, the player always receives the highest possible bonus they deserve, but they can only get one bonus. Note that this means that some bonuses are skipped, if a player earns enough points on a single level. Also, once a bonus has been earned, all future bonuses have to be for achieving strictly higher bonus levels.

Consider the following example where $c = 100$ and a player has played 6 levels, scoring 50, 130, 600, 200, 450 and 1000 on those levels, respectively. Here are the player's scores after each level after adding bonuses:

Level 1: 50, since there is no bonus
Level 2: 50 + 130 + 50 = 230, the bonus is 50 since the total score was at least 100 but not 200. Notice that even though this bonus puts the player over 200, they don't subsequently earn the bonus for 200 total points. Only one bonus can be awarded at the end of each level.
Level 3: 230 + 600 + 400 = 1230, the bonus is 400 for clearing 800. Notice that due to the relatively high score on this level, the 100 and 200 bonuses were skipped.
Level 4: 1230 + 200 = 1430, there is no bonus since 1600 hasn't been cleared yet.
Level 5: 1430 + 450 + 800 = 2680, since 1600 was now cleared.
Level 6: 2680 + 1000 + 1600 = 5280, since 3200 was now cleared.

The Problem

Given the number of levels in Geo's video game, the value of *c* for that video game as described above, and a player's scores on each of the levels of the game, calculate the player's actual total score, including bonuses.

The Input

The first line of input will consist of a single positive integer, *v (v ≤ 100)*, representing the number of input cases to process. Input for each case follows, one case taking two lines.

The first line of input for each case contains two space-separated positive integers, n ($n \leq 100$), and c ($10 \leq c \leq 1000$, c is even), representing the number of levels of the video game and the value of c as described above for the game, respectively. The following line contains n space separated integers, each in between 0 and 100000 inclusive, representing the player's score on each of the n levels, in order.

The Output

For each input case, output a single integer on a line by itself representing the player's total score for that game, including bonuses.

Sample Input	**Sample Output**
2 6 100 50 130 600 200 450 1000 5 10 319 77 929 13 100000	5280 143178

Counting Factors

Filename: *counting*
Time Limit: *2 seconds*

Viraj really wants to beat Spencer at this year's FHSPS. He decided that his best shot was to sneak in a number theory problem into the set, since mathematics is one of his strengths. Lucky for Viraj, Arup (problem setter) encrypted his files with an insecure cipher, which Viraj easily broke. With access to all of Arup's files, planting a problem in the set was child's play. Viraj likes prime numbers very much, but only ones that are less than one hundred. We call these numbers Viraj primes. A Viraj prime factorization is simply a prime factorization where all the primes are less than one hundred. Viraj is curious: how many different integers with Viraj prime factorizations have precisely *n* factors?

Write a program to answer his query.

The Problem

Given a positive integer, *n*, how many integers with only prime factors less than 100 have exactly *n* factors. Since this number could be very large, calculate it modulo $10^9 + 7$.

The Input

The first line of input will consist of a single positive integer, *c (c ≤ 100)*, representing the number of input cases to process. Each of the next *c* lines will contain a single positive integer, *n* ($n \leq 10000$), the value for the input case.

The Output

For each input case, output a single integer on a line by itself representing the number of integers with prime factors less than 100 only that have exactly *n* divisors, mod $10^9 + 7$.

Sample Input	Sample Output
4	1
1	25
2	25
3	325
4	

Defeating Dragons

Filename: *dragons*
Time Limit: *5 seconds*

You are playing a video game where you must kill all dragons that come your way. Every dragon has precisely two weaknesses: one poison which kills them, and one weapon which kills them. Your character is limited in the number of items (poisons and weapons) he can carry.

The Problem

Given a list of each dragon you must vanquish, and the poison and weapon that can kill it, determine the fewest number of items (number of poisons and number of weapons) you must carry with you to kill all dragons.

The Input

The first line of input will consist of a single positive integer, *c* ($c \leq 50$), representing the number of input cases to process. The input cases follow. The first line of each input case contains a single positive integer, *n* ($n \leq 1000$), representing the number of monsters you must kill for that case. The following *n* lines each contain three space-separated strings (in between 1 and 20 lower case letters, inclusive), representing the name of a monster, the poison that kills that monster, and the weapon that kills that monster, respectively. The name of all weapons will be distinct from the name of all poisons, and all monster names will be distinct within a single input case.

The Output

For each input case, output a single integer on a line by itself representing the minimum number of items necessary to kill all dragons for that input case.

Sample Input

```
2
4
dragona water sword
chewy acid gun
superrodent water gun
lochness acid cannon
5
dragona arsenic pistol
dragonb acid sword
dragonc water pistol
dragond water pistol
dragone arsenic pistol
```

Sample Output

```
2
2
```

Electric Car Race

Filename: *electric*
Time Limit: *20 seconds*

The electric car race involves starting at a home base, visiting each of several check points exactly once, and returning to the home base. The catch is that only some of the check points have charging stations, so depending on what order you visit the check points, you may end up running out of charge before getting back to home base.

You start out with a full charge at home base, and any time you visit a check point with a charging station, you fully charge your car before moving onto the next check point.

Your friend Pamela has decided that she doesn't want to think about what order to visit the check points and she's simply going to visit them in a random order and hope for the best. In particular, at each step, she simply picks at random one of the check points she has yet to visit, to ensure that none get visited more than once. Once all are visited, she heads back to home base.

What is the probability that Pamela will succeed in visiting all of the check points and returning to home base without running out of a charge?

The Problem

Given a list of check points, the distances between pairs of check points and home base and each check point, as well as how far Pamela's car can go on a single charge, determine the probability that Pamela will be able to visit each of the check points and return to home base successfully.

The Input

The first line of input will consist of a single positive integer, c *(*$c \leq 20$*)*, representing the number of input cases to process. The first line of each input case will contain three space separated integers, n ($1 \leq n \leq 10$), the number of check points for that input case, m ($0 \leq m \leq n$), the number of those check points that have charging stations, and d ($1 \leq d \leq 500$), the distance in miles the car can travel for the input case on a full charge before needing to charge again.

The next line contains n+1 space separated non-negative integers. The first of these is 0, the distance from home base to itself. The rest of these are the distance from home base to each of the check points, respectively. Of the lines that follow, the i^{th} line ($1 \leq i \leq n$) contains distances from check point i to home base (first integer listed), and each of the other check points, respectively. The first m check points listed will be the ones with the charging stations. Note that the distance from location a to location b might be different than the distance from location b to location a. All of the $(n+1)^2$ distances in this part of the input will be less than or equal to 1000.

The Output

For each input case, on a line by itself, output the probability that Pamela will successfully visit each check point and return back to home base. Output the probability in the form of a fraction

```
X/Y
```

where Y is positive and X and Y share no common factors.

Sample Input

```
2
3 1 100
0 50 100 150
60 0 20 125
80 60 0 70
10 100 50 0
4 2 100
0 30 40 50 60
25 0 15 80 90
33 43 0 53 43
44 33 55 0 66
15 35 20 50 0
```

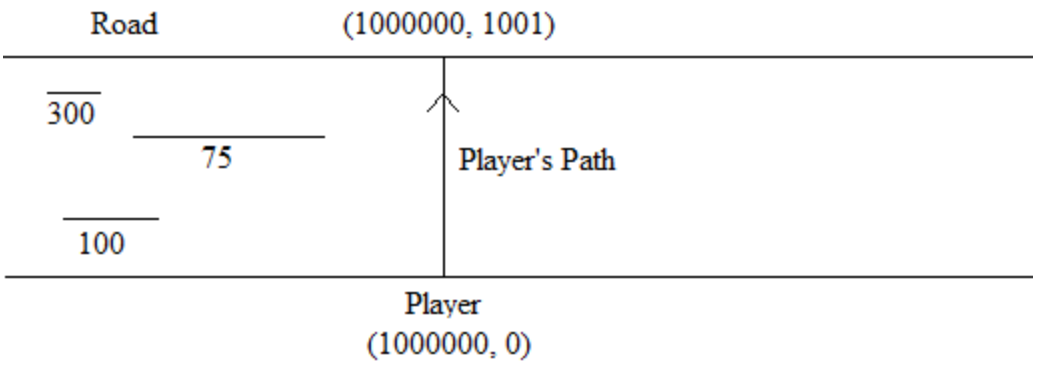
Sample Output

```
1/6
1/8
```

Frogger for Four Year Olds

Filename: *frogger*
Time Limit: *15 seconds*

Arup (problem setter) has a four year old daughter, Anya. He would really like to get Anya into video games, but her skills are quite limited. She can press buttons every now and then, but that's about it. Arup has devised a game that she might be able to play, based on the old video game Frogger. In the game, cars are crossing the screen from left to right and the player has to control their frog to cross the road. In the regular game, the frog can move in all four directions: up, down, left and right to avoid cars. Since planning moves in different directions would be too difficult for Anya, Arup's game involves a single button push. Also, since running into a car would be traumatic for Anya, even if it was just in a game, Arup has replaced the cars with moving prizes that you are supposed to catch (intersect with)! Thus, the goal of the game is to obtain a set of prizes with a maximal sum of points. When Anya presses the button, it moves her piece, which can be modeled by a single point, at a constant speed, vertically up the screen. She obtains whichever prizes her piece intersects with. Each piece will be modeled as a horizontal line with an initial position and all pieces will be moving at the same constant speed from left to right, though this speed might be different than the speed the player moves. Each prize will have an associated point value and the goal of the game is to maximize the sum of the point values of each prize obtained.



For example, if the velocities of each of the prizes was very slow compared to the player's velocity, in the picture above, the player could wait to press the button until the player's path would intersect with the prize worth 100 and the prize worth 300, for a total score of 400.

For the purposes of this problem, we model the playing area in the Cartesian plane with the player's initial position set at (1000000, 0). The initial positions and velocities of each of the prizes will be such that if the player were to press the button at the first possible opportunity, no prize would pass her by.

The Problem

Given the velocity of Anya's piece, the velocity of all of the prizes, initial position of all of the prizes, the length of each of the prizes and the number of points each prize is worth, determine the maximum amount of points Anya can earn by pressing the button at the appropriate time.

The Input

The first line of input will consist of a single positive integer, c ($c \leq 10$), representing the number of input cases to process. The input cases will follow.

The first line of each input case will have three space separated positive integers: n ($n \leq 100000$), v ($v \leq 100$), and w ($w \leq 100$), representing the number of prizes, Anya's velocity in units/sec and each prize's velocity in units/sec, respectively.

The following n lines will describe the prizes, one line per prize. Each of these lines will contain four positive integers: x ($x < 1000000$), y ($y \leq 1000$), l ($l \leq 1000$), and p ($p \leq 10000$), separated by spaces, representing the initial x and y coordinates of the left end of the prize (in units), the length of the prize (in units), and the value of the prize, respectively. The input will be such that the duration of time when a button press can obtain the maximal score will be at least one consecutive interval of 0.001 or more seconds.

The Output

For each input case, output a single integer on a line by itself representing the maximum score Anya could obtain by pressing the button at the appropriate time.

Sample Input

```
2
3 100 2
100 100 200 100
50 300 100 300
250 220 500 75
5 1 1
5 1 1 1000
4 2 2 900
3 3 3 800
2 4 4 200
1 5 5 100
```

Sample Output

```
400
3000
```

Ground Game

Filename: *game*
Time Limit: *1 second*

In the game Ground Game, a player can press one of four keys:

`>`	to move right (greater than)
`<`	to move left (less than)
`^`	to move up (carrot)
`v`	to move down (lowercase v)

At the beginning of the game, a player starts at the ground level. When he moves left or right, he stays at the same level. If he moves down, he moves underground one more level. If he moves up, he moves one level up towards the ground. He can never move higher than the ground level and never tries to do so, but he can move at the ground level as much as he likes.

The Problem

Given a full list of key presses by a player, determine the maximum number of levels below ground he moved during the game.

The Input

The first line of input will consist of a single positive integer, *n (n ≤ 100)*, representing the number of input cases to process. The input cases follow, one per line. Each input case is a string of in between 1 and 100, characters, each from the set `{'>', '<', '^', 'v'}`. It is guaranteed that no input string will give directions that would move the player higher than the ground level.

The Output

For each input case, output a single integer on a line by itself representing the maximum number of levels below the ground level the player for the input case traveled in the game.

Sample Input	**Sample Output**
2	7
>>>vvvvvv<<^^>vvv^^>>>	0
>>>><<<<<>>>>><	

High Score

Filename: *highscore*
Time Limit: *3 seconds*

A friend wants you to help her with the score board for her new video game. She wants you to provide a ranked score list of players. She wants the players sorted in a rather interesting way. Her game has multiple levels and it's always better to achieve a higher level. Any player who has gotten to level x should be ranked above any player who has gotten to y, given that $x > y$. If two players have reached the same level, then their ranking should be based on total score - whoever has the higher total score should come first in the ranked list. But, if two players have reached the same level AND gotten the exact same score, then, the tie is broken based on whichever player outscored the other player in the earliest level where their scores were different. Finally, if some players' scores are identical on every level, then they should be ranked by their initials, in alphabetical order.

The Problem

Given a list of scores for each player on your friend's video game, output a ranked list based on the specification given above.

The Input

The first line of input will consist of a single positive integer, c ($c \leq 100$), representing the number of input cases to process. The first line of each input case will contain a single positive integer, n ($n \leq 100$), the number of players for that input case. The following n lines will contain information about each player, one line per player. Each of these lines has the following space separated items: first, a player's initials (3 uppercase letters), followed by k ($k \leq 100$), representing the level the player achieved. This is followed by k non-negative integers, s_1, s_2, s_3, ..., s_k, respectively, where s_i ($s_i \leq 10^9$) represents the score on level i for that player. The scores will be such that even with all bonuses added in, all players' total scores will fit into a 32 bit signed integer. All players' initials within a single input case will be unique.

The Output

For each input case, output a case header with the following format: `Game #g`

where g is the 1-based input case number. This should be followed by n lines, each containing the initials for one player, in sorted order, as specified by the problem.

Sample Input

```
2
3
FTA 3 300 400 500
ZZZ 2 100 2000
ARG 3 400 500 200
4
ZZZ 3 500 501 499
DEG 3 500 500 500
CZZ 3 500 500 500
DEF 3 500 500 500
```

Sample Output

```
Game #1
FTA
ARG
ZZZ
Game #2
ZZZ
CZZ
DEF
DEG
```

Igloo Construction

Filename: *igloo*
Time Limit: *1 second*

One of your friends has a bizarre idea for a game: building igloos! The goal of the game is to construct an igloo with a given volume of ice. For the purposes of this problem, an igloo is half of a sphere with a smaller half sphere carved out. Depending on where the igloo is built, there is a minimum requirement for the thickness of the igloo. Show your friend that this game is silly by writing a computer program that computes the maximum outer radius of an igloo that can be built, given the total volume of ice available and the necessary thickness of the igloo.

The Problem

Given a volume of ice available in ft^3 and the minimum necessary thickness of the igloo in feet, determine the maximum possible outer radius of an igloo that can be built, in feet.

The Input

The first line of input will consist of a single positive integer, *c (c ≤ 100)*, representing the number of input cases to process. The input cases follow. Each input case, one per line, consists of two space separated positive integers, *v* ($v \leq 1000000$) and *t* ($t \leq 10$), representing the volume of ice available for building the igloo (in ft^3) and the minimum thickness required for the igloo (in ft). It is guaranteed that $v > \frac{2}{3}\pi t^3$, meaning that for each input case, an igloo with the required thickness can be built with the ice available.

The Output

For each input case, output a single real number rounded to three decimal places, representing the maximum outer radius possible for building an igloo with the required thickness with the ice available.

Sample Input	Sample Output
2 1000 2 12345 3	9.902 27.077

Just in Case (the set is finished...)

Filename: *just*
Time Limit: *15 seconds*

Initially Arup was upset that Viraj cracked his cipher so easily. But once Arup read Viraj's problem, he loved it so much. Mostly he was jealous that Viraj made up such a nice problem. Last year, the FHSPS Playoff set was solved so quickly. This year, Arup wanted to make sure it wasn't solved quite so fast. After thinking carefully about Viraj's problem, he decided that he could probably solve it for a larger bound on n. So, Arup decided that, just in case the other 9 problem in the set were solved, he'd prepare a tenth - a version of Viraj's question with different bounds! After all, Arup's curiosity ventures further than 10^4. In fact, Arup wants to ask the following question:

How many different integers with Viraj prime factorizations have precisely *n* factors?

where n can be up to 10^{10}.

Write a program to answer his query.

The Problem

Given a positive integer, *n*, how many integers with only prime factors less than 100 have exactly *n* factors. Since this number could be very large, report it modulo $10^9 + 7$.

The Input

The first line of input will consist of a single positive integer, *c (c ≤ 60)*, representing the number of input cases to process. Each of the next *c* lines will contain a single positive integer, *n* ($n \leq 10^{10}$), the value for the input case.

The Output

For each input case, output a single integer on a line by itself representing the number of integers with only prime factors less than 100 that have exactly *n* factors, mod $10^9 + 7$.

Sample Input	Sample Output
4	1
1	25
2	25
3	325
4	