A. Frogs

7 points

Last year, in the Philomath frog jumping contest, the poor orphan boy, little Timmy, lost by a single frog width. That is, his frog jumped and landed just shy of the finish line, when Bluto's frog hopped just beyond the finish line to win. Everyone felt so bad that they want something done, but they don't know what.

Delores, the high school Biology teacher, discovered that individual frogs always jump the same distance. The real trick to the frog jumping contest is making them jump faster, not further. So, the people want a finish line that is an even multiple of the distance little Timmy's frog will jump, so he won't look at everyone with those sad eyes if his frog jumps just shy of the finish line. However, Delores feels that although Bluto is big for his age and smells bad, he should not be disadvantaged either. So, she wants a program that takes in the distance that every frog jumps, and computes a length of course that is an even multiple of all of their jump lengths. In particular, she wants the smallest, non-zero length that qualifies.

Delores dual majored in Math and Biology, so she reminds you of the Euclidean Algorithm for finding GCD's because she is pretty sure you need it.

$$\gcd(a,b) = \begin{cases} a & b = 0 \\ \gcd(b, a \bmod b) & b \neq 0 \end{cases}$$

Input

The first line is a number $0 < n < 1000$ which tells you how many problems follow. The line containing each problem begins with a number $1 < m < 1000$ which tells you how many frogs are in the contest. This is followed by m integers that tell you how many centimeters each frog jumps. A frog cannot jump more than 35 feet.

Output

For each problem, you should output the length, in centimeters, of the track that is the shortest, non-zero length that is evenly divisible by all of the frogs' jumping length.

Sample Data

Input	Output
3 4 12 24 36 4 3 3 5 7 8 4 5 10 12 22 12 6 8	72 105 1320

B. Cannonballs

7 points

It's the zombie apocalypse! It's generally agreed that the best place to be in this event is a mall, due to the presence of food and potential weaponry, as well as an abundance of doors that one can close. Unfortunately, you and your friends instead find yourselves trapped in a Civil War museum! You find that all the muskets have been filled with cement, and thus are useless as weapons. The cannons, however, are perfectly functional, and your only hope against the zombie horde.

Now, you need to know how many cannonballs you have, in order to best plan your next move, and ensure that your rescue helicopter shows up at exactly the right time (it would be less dramatic if it showed up while you still had ammunition). But the horde is approaching, and you don't have time to count each ball! Luckily, these balls are already grouped into square pyramids, as they did during the Civil War. Knowing the height of each pyramid, and the number of pyramids, how many cannonballs can you fire until becoming completely overrun?

Input

The input will begin with a single integer $0<n<1000$, indicating the number of problems to follow. This will be followed by n lines. Each line will start with $0<m<50$, the number of pyramids, followed by m numbers $0<h_i<500$ indicating the height of each pyramid. Each pyramid starts with a base of h_i^2 with each layer having one fewer cannonball to each side than the previous layer.

Output

For each line, output the total number of cannonballs that you can fire at the zombie horde.

Sample Data

Input	Output
4 1 1 5 3 4 5 4 3 1 2 10 2 2 2 2 2 2 2 2 2 2	1 143 5 50

C1. Cable

10 points

The Comcash Cable Co. has won the exclusive contract with the city of Eugene to provide cable services. The satellite uplink stations are very expensive, so they want to provide a single network of cable along each street. Also, the cable splitters are expensive, so CCC will only pay for one splitter for each house. You can, however, split the cable as many times as needed at each house. Obviously, they would like to use as little cable as possible, so they have hired you to compute the amount of cable needed. Given these constraints, you can think of this problem as the minimum length of cable needed to connect the houses into a single cable network.

In Eugene, the odd addresses are on the North or East side of the street. The even addresses are on the South or West side of the street. Each address represents the distance in feet from the center of the property to Willamette Street or the Willamette River, depending on whether the street runs East-West or North-South. City regulations require that when you string cable across the street, it must go straight across the street (i.e. perpendicular to the flow of traffic). Each street is 30 feet across. Thus, the shortest direct path between addresses 1245 and 1350 is 135 feet: 105 feet down the street, plus 30 feet to go across the street.

Input

The first line of input is a number $0 < n < 1000$ that tells you how many streets will follow. This is followed by n lines of street data. The line of data corresponding to each street consists of a number $0 < m < 100$ followed by m space delineated addresses of cable subscribers.

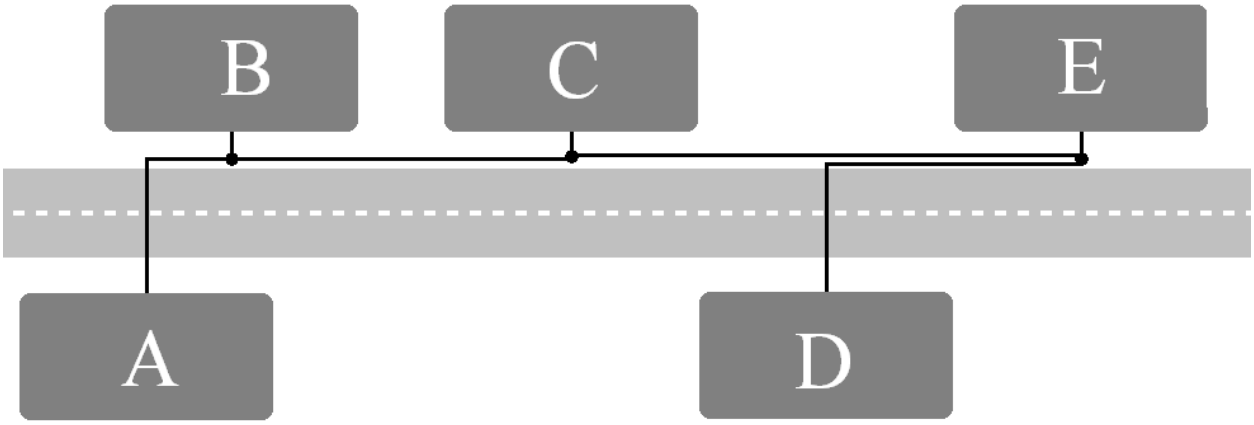
Output

Your program should output a line for each street in the input. Your output should be how many feet of cable are required to connect the addresses on that street.

Sample Data

Input	Output
3 5 100 101 250 195 280 5 1031 1055 1085 1175 1176 5 21134 21196 21155 21135 21075	240 feet 175 feet 173 feet

Notice that in the following picture, the cable stretches between houses C and E and another cable runs between houses D and E. You cannot split the cable halfway between C and E because then you would have more splitters than allotted, even though this would decrease the amount of cable.



C2. Satellites

10 points

As in the previous problem, the Comcash Cable Co. wants you to figure out the least amount of cable that will feed each street. However, they have added a price structure that allows certain subscribers to be preferred subscribers. A preferred subscriber has priority on the cable. That is, when the preferred subscriber's packets are en route, all other subscribers' packets must wait. Obviously, there can be only one preferred subscriber on each cable run. So, although satellite uplink stations are expensive, CCC must pay for one at each preferred subscriber's house.

The output for this problem is the same as before, but the input is slightly different. You don not need to worry about the case where there are no preferred subscribers, because they will use your algorithm from the previous question to figure out that street. You do not need to worry about a street with all preferred providers, because they will all have uplink stations.

Input

The first line of input is a number $0 < n < 1000$ that tells you how many streets will follow. This is followed by $2n$ lines of street data, with each street requiring 2 lines of data. The first line of data corresponding to each street consists of a number $0 < i < 100$ followed by i space delineated addresses of regular cable subscribers. The second line of data corresponding to each street consists of a number $0 < j < 100$ followed by j space delineated addresses of preferred cable subscribers.

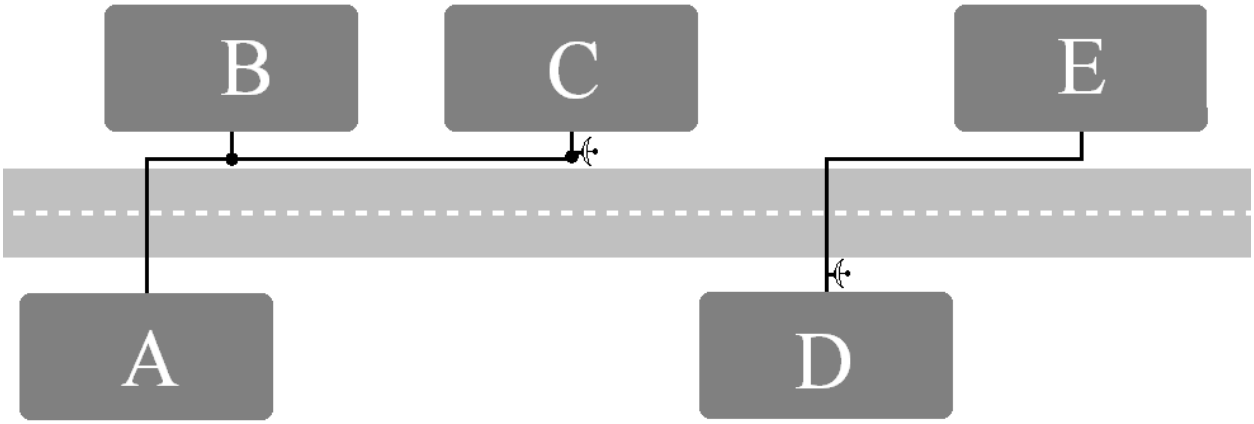
Output

Your program should output a line for each street in the input. Your output should be how many feet of cable are required to connect the addresses on that street.

Sample Data

Input	Output
3 3 101 250 195 2 100 280 1 1176 4 1031 1055 1085 1175 5 1134 1196 1155 1135 1075 1 1197	146 feet 31 feet 184 feet

The following picture shows a possible configuration if houses C and D are preferred subscribers. Notice that the uplink stations are located at C and D, and no cable connects them.



D. Robots

10 points

The Company has designed a self-replicating robot. After some allotted period of time, the robot creates two exact duplicates of itself. Each robot randomly selects a name for each of it's children from a list of names. The problem is that there is the possibility of duplicate names and The Company has contracted you to implement and automated solution.

The Company wants each robot to have a last name consisting of the first letter of its parent's first name, followed by the first letter of his grandparent's first names, etc. until you get to the root or original robot. The original robot will not have a last name. You can assume that each name will consist only of letters.

Input

The input will begin with a line consisting of an integer $0<n<1000$ followed by n lines representing n problems. Each problem will begin with an integer $0<k\leq 512$ followed by k names of each robot representing the binary family tree in a level order fashion. You can assume $k = 2^m$ for $0\leq m\leq 9$.

Output

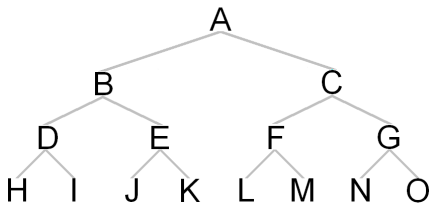
For each problem, you should output k lines with the name of each robot, first and last name, in the order they were given to you. The first letter of each first and last name should be capitalized.

Sample Data

Input	Output
2 3 Peter Paul Mary 7 Happy Sneezy Bashful Grumpy Dopey Sleepy Doc	Peter Paul P Mary P Happy Sneezy H Bashful H Grumpy Sh Dopey Sh Sleepy Bh Doc Bh

Level Order for a Complete Binary Tree

The level order for a binary tree is a list of nodes at each level from left to right. For example, the level order list of nodes A, B, C, D, E, F, G, H, I, J, K, L, M, N, O corresponds to the following tree.



E. Genes

12 points

Bioinformatics is the branch of life science that deals with the study of application of information technology to the field of molecular biology. Common activities in bioinformatics include mapping and analyzing DNA and protein sequences, aligning different DNA and protein sequences to compare them and creating and viewing 3-D models of protein structures.

The primary goal of bioinformatics is to increase our understanding of biological processes. What sets it apart from other approaches, however, is its focus on developing and applying computationally intensive techniques (e.g., data mining, and machine learning algorithms) to achieve this goal. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, genome-wide association studies and the modeling of evolution.

Your task is to examine two strings of DNA, the base string and the desired mutation. By adding bases and changing bases, you will return the minimum number of changes (not including additions) to the base string.

Input

The first line is a number $0 < n < 1000$ which tells you how many problems follow. Each problem consists of two lines. Each line of the problem consists of a number $1 < m < 1000$ followed by m characters of `A`, `C`, `T` or `G` with no spaces or other characters between them. The first line is a base sequence of genetic data. The second line is the target mutation. The base sequence will never be longer than the target mutation.

Output

For each problem, you should output the minimum number of changes required to turn the base sequence into the desired mutation. Turning the base sequence into the desired mutation may also involve adding letters to the base, but you do not count those additions in your result count.

Sample Data

Input	Output
4 6 GATAGT 6 GCATAA 6 GGTTAA 10 GGTTAATTTA 3 CAT 5 GATAC 6 GATACA 10 GTTCAAGGGA	5 0 1 2

A related problem is the Longest Common Subsequence problem in which we want to know the number of characters that two strings have in common in the same order. To solve the LCS problem, we just consider deleting a letter from the first sequence or the second sequence, and report whichever choice yields the maximum length of undeleted letters. Here is the pseudo-code for that problem:

```
LCS(A, B):
     if len(A) == 0
          return len(B)
     else if len(B) == 0
          return len(A)
     else if A[0] == B[0]
          return 1 + LCS(A[1..end],B[1..end])
     else
          return max{LCS(A[1..end],B), LCS(A,B[1..end])}
```

This might compute the same codes more than once because in looking at two strings, we will consider the sequence `LCS(A[1..end],B), LCS(A[1..end],B[1..end]`]) and we also consider the sequence `LCS(A,B[1..end]), LCS(A[1..end],B[1..end]`]). This means the algorithm has exponential timing. To fix this, we simply add a few lines that cause us not to recompute:

```
LCS(A, B):
     if known(len(A),len(B))
          return known(len(A),len(B))
     else if len(A) == 0
          result = len(B)
     else if len(B) == 0
          result = len(A)
     else if A[0] == B[0]
          result = 1 + LCS(A[1..end],B[1..end])
     else
          result = max{LCS(A[1..end],B), LCS(A,B[1..end])}
     known(len(A),len(B)) = result
     return result
```

This algorithm will have much quicker running time.

F. Races

12 points

There was a race held yesterday. All of the times for the participants were recorded digitally when the participant crossed the finish line. Before the race started, some times were recorded separately to make sure the time recorder was working. Unfortunately, some carelessness with the digital data resulted in the two lists of times being combined. They were combined such that the real race times are still in the same relative order, with the text times randomly placed in the input. You don't know whether the race times were in increasing or decreasing order originally.

Your job is to try to recover the real list of race times. You can assume the real list is the longest increasing or decreasing sequence of numbers in the input. All the other numbers are just test times. For example, if you saw "15 19 9 26 7 14 32 18 47 51 17", you would conclude that the race times were 15 19 26 32 47 and 51, because it is the longest increasing or decreasing sequence embedded in the list. The sequence "9 14 18 47 51" is also increasing, but not as long. The sequence "15 9 7" is decreasing, but also not as long. Some of the runners may have finished at the same time, so consecutive numbers that are equal may also be part of the race timings.

Input

The first line is a number $0 < n < 1000$ which tells you how many problems follow. The line containing each problem begins with a number $1 < m < 100$ followed by m positive integers less than 10,000 representing either seconds each runner took to finish, or superfluous test data.

Output

For each problem, you should output the largest sequence of numbers that either don't increase or don't decrease in order in the input data.

Sample Data

Input	Output
4 11 1 2 57 43 3 43 42 37 4 32 17 11 17 32 60 59 37 42 58 57 43 43 57 11 1 2 3 4 57 43 43 42 37 32 17 7 57 43 43 42 37 32 17	57 43 43 42 37 32 17 17 32 37 42 43 43 57 57 43 43 42 37 32 17 57 43 43 42 37 32 17