Problem Set 2012

This problem set contains 10 questions (A–J) in 19 pages.

Hosted by
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA Malaysia

In collaboration with
Kulliyyah of Information and Communication Technology
International Islamic University Malaysia

3–4 November 2012

A Anti-Arithmetic Permutation

Input	Standard Input
Output	Standard Output
Time Limit	4 seconds

Problem Description

We call a permutation p_0, p_1, ... , p_{n-1} of integers 0, 1, ... , n-1 anti-arithmetic, when there are no three-term arithmetic series in this permutation, i.e. there are no such three indices $i < j < k$, that integers p_i, p_j, p_k make an arithmetic series. For example, the series of integers 3, 1, 0, 4, 2 is an anti-arithmetic permutation of integers 0, 1, 2, 3, 4. The series 0, 5, 4, 3, 1, 2 is not an anti-arithmetic permutation, because its first, fifth and sixth term: 0, 1, 2 form an arithmetic series (as well as its second, fourth and fifth term: 5, 3, 1 and second third and fourth term: 5, 4, 3 form arithmetic series). Given a permutation of length n determine whether the given permutation is anti-arithmetic or not.

Input

Input starts with an integer ***T***, the number of test cases.

Each test case consists of two lines. First line contains an integer n. Next line contains n integers separated by a single space. These n integers denotes a permutation of 0, 1, .., n-1. n is between 3 and 50 inclusive.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is "YES" when the given permutation is anti-arithmetic or "NO" otherwise. Quotes are for clarity only.

Sample input and output is on the next page.

Sample Input	Sample Output
12 4 3 1 0 2 9 0 8 4 6 2 3 7 5 1 7 1 5 3 2 6 4 0 6 3 2 5 1 4 0 10 0 8 4 2 6 9 1 5 3 7 6 3 1 5 2 4 0 3 1 0 2 3 2 0 1 5 0 4 2 1 3 7 1 5 3 0 4 2 6 6 2 0 4 5 1 3 10 4 3 1 6 9 2 5 8 0 7	Case #1: YES Case #2: YES Case #3: YES Case #4: NO Case #5: YES Case #6: YES Case #7: YES Case #8: YES Case #9: YES Case #10: YES Case #11: YES Case #12: NO

B Longest Balanced Sub-Sequence

Input	Standard Input
Output	Standard Output
Time Limit	1 seconds

Problem Description

We call a sequence of numbers **balanced**, if the amount of positive values in the sequence equals the amount of negative values.

Input

The first line of the input contains an integer T, $1 \leq T \leq 15$, indicating the number of test cases.

For each test case, two lines appear. The first line contains an integer N, $0 \leq N \leq 100000$. The second line contains N **non-zero** 32-bit signed integers.

Output

For each test case, the output contains a line in the format Case #x: M, where x is the case number (starting from 1) and M is the length of the **Longest Balanced Sub-Sequence (LBSS)** of consecutive elements.

Sample input and output is on the next page.

Sample Input	Sample Output
2 8 -1 -5 1 -7 8 -6 -9 -2 17 5 -2 1 3 7 9 -9 -1 6 -7 -1 2 8 3 1 -2 -1	Case #1: 4 Case #2: 12

Explanation

For the first test case, the LBSS is sub-sequence (-5 1 -7 8) or (1 -7 8 -6); both with two positive and two negative values.

For the second test case, the LBSS is sub-sequence (9 -9 -1 6 -7 -1 2 8 3 1 -2 -1) with six positive and six negative values.

C CHARITY BOOTH RENTAL

Input	Standard Input
Output	Standard Output
Time Limit	3 seconds

Problem Description

The Kajang Municipal Council is organising a fundraising event to raise money for charity. The council plans to provide booth spaces for rent to government agencies (***GA***), private companies (***PC***) and the general public (***GP***) who wish to participate in that event. The organiser has decided to allocate booth spaces as follows:

0 < booth spaces for ***GP*** < booth spaces for ***GA*** < booth spaces for ***PC***

As an example, for 10 booth spaces, the following allocations are possible

GP =1, ***GA*** =2, ***PC*** =7
GP =1, ***GA*** =3, ***PC*** =6
GP =1, ***GA*** =4, ***PC*** =5
GP =2, ***GA*** =3, ***PC*** =5

Write a program to calculate the number of possible allocations given the number of booth spaces.

Input

Each case is the number of booth spaces T where 6 <= ***T*** <= 1000000. The input is terminated with a 0.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the number of booth spaces and the answer with a colon (':') separating them. The output for each case must be printed on a separate line.

Sample Input	Sample Output
8 10 0	Case #1: 8:2 Case #2: 10:4

D FIND THE MARBLES

Input	Standard Input
Output	Standard Output
Time Limit	3 seconds

Problem Description

A group of friends are playing "Find the Marbles" game. A handful of marbles are thrown on the ground. The position of the marbles is represented by integer coordinates in a two-dimensional space. The players can only walk on a straight line to find the marbles. The person who can find the highest number of marbles on a straight line wins the game. You are to help them find the number of marbles that the winner found.

Input

The first line of the input is a single positive integer ***T*** indicating the number of test cases. Each case starts with the number of marbles ***N***, where $1 < N < 100$. On each of the following *N* lines there are a pair of integers separated by a blank that represents the coordinate in a two-dimensional space. No pair will occur twice.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is an integer representing the largest number of marbles that lie on one line.

Sample input and output is on the next page.

Sample Input	Sample Output
2 5 1 1 2 2 3 3 5 10 6 11 3 4 5 6 7 8 8	Case #1: 3 Case #2: 2

E Supercap Travels

Input	Standard Input
Output	Standard Output
Time Limit	4 seconds

Problem Description

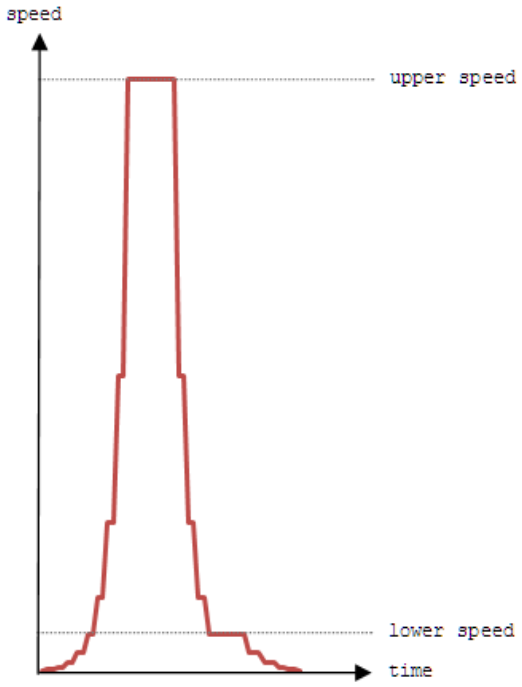
In the year 2037, i.e., two and a quarter centuries after the first commercial rail road locomotive was built, the world triumphed with the successful roll-out of supersonic magnetic levitation capsules called supercaps. These supercaps are the new generation of Transrapid vehicles, which can readily attain and travel at top speed of 512 ms^{-1}. As a result, travellers can make trips between cities in matter of seconds if not minutes.

The power of the supercap lies in its ability to accelerate and decelerate instantly. The acceleration is strictly in integral powers of two starting with the zeroth power such that each subsequent speed is double the one before. Each speed during acceleration is maintained for a single second only. For example the speed during the first, second, third and fourth seconds are 2^0 ms^{-1}, 2^1 ms^{-1}, 2^2 ms^{-1}, 2^3 ms^{-1}. The acceleration continues in this manner until a computed *upper speed* ($\leq$ 512 ms^{-1}) is reached. For sake of simplicity, you can assume that the supercap's acceleration is instantaneous and discrete.

The supercap decelerates from its *upper speed* to a predetermined *lower speed* (= 16 ms^{-1}) also strictly in integral powers of two, such that each subsequent speed is half the one before. Similar to acceleration, each speed during deceleration to the *lower speed* is maintained for a single second only. You can also assume that the deceleration is instantaneous and discrete.

However, the supercap has been designed to glide, i.e., maintain a speed for duration longer than one second, at the *upper speed*, as well as, at the *lower speed* and the speeds below it. In this way, the supercap can slow down gradually to halt, if necessary.

In summing, the motion of the supercap can be divided into four parts. In the first part, the supercap will accelerate multiplicatively to a computed *upper speed*. In the second part, the supercap glides at the *upper speed*. In the third part, the supercap decelerates multiplicatively to a predetermined *lower speed*. In the final part, the supercap slows down to halt by gliding at the *lower speed* and/ or the speeds below it. The speed-time graph below illustrates the typical motion of a supercap.



During the first phase of development, the supercap transrapid rail road links will be developed to connect the capital city of each region to only one city in each of its neighbouring regions. Cities within a region will not be connected by the supercap transrapid system.

The cost of developing the supercap transrapid system will be borne by the travellers. The fare is relative to the speed of the supercap. It will be based on the travelling time above, at, and below the *lower speed*. The supercap transrapid system corporation has come up with a simple formula to calculate the fare ***F***, which is proportional to result of the equation below. It is profitable to the corporation if a supercap's time of travel above the *lower speed* A is as long as possible and its time of travel at or below the *lower speed* B is as short as possible. The penalty for ***B*** is 100 times that of ***A***.

$$F \propto (A - 100B)$$

where A = Travelling time > lower speed

and B = Travelling time ≤ lower speed

You are required to determine the most profitable supercap rail road links that a region must develop first, i.e., the regional links from which the supercap transrapid system corporation can derive maximum fare revenue.

Input

The first line of input contains an integer ***T*** ($1 \leq \boldsymbol{T} \leq 10$), the number of test cases. On the first line of each test case, there will be an integer ***R*** ($1 \leq \boldsymbol{R} \leq 10$), the number of neighbouring regions, followed by information about the cities with Transrapid stations in each of the ***R*** neighbouring regions.

Each neighbouring region information starts with an integer ***C*** ($1 \leq \boldsymbol{C} \leq 10$), the number of cities in that region followed by ***C*** lines of city-distance description. Each city-distance description consists of a city name ***Y*** and an integer D ($1{,}000 \leq \boldsymbol{D} \leq 2{,}000{,}000$), the distance between the Transrapid station in capital city ***Z*** to the Transrapid station in city ***Y***. The city name is a single word and the distance is specified in meters. You can assume that the distances between the Transrapid stations are unique.

Output

The output comprise of one line for each test case set. The line begins with the prefix "Case #x:" where x represents the case number (starting from one and incrementing at each new test case), followed by a single space, and then the names of the cities to which the rail road links must be developed to satisfy the conditions of the project. The order of the names in the output will follow the order of the neighbouring regions in the test case, and will be separated by a single space.

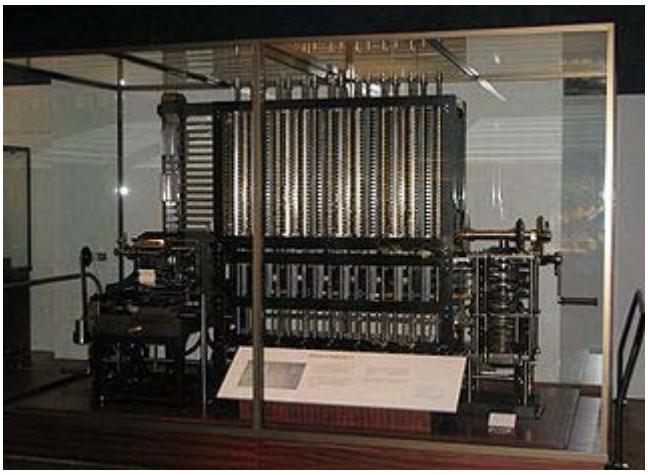
Sample Input	Sample Output
3 1 2 Yoyo 8712 Zing 3118 2 1 Burj 18000 3 Dino 92400 Ambro 47624 Ebb 25725 1 2 Klm 1500000 Mys 1450000	Case #1: Yoyo Case #2: Burj Ambro Case #2: Mys

F Charles Babbage's Difference Engine

Input	Standard Input
Output	Standard Output
Time Limit	3 seconds

Problem Description

A difference engine is an automatic mechanical calculator designed to tabulate polynomial functions. The name derives from the method of divided differences, a way to interpolate or tabulate functions by using a small set of polynomial coefficients. Both logarithmic and trigonometric functions, functions commonly used by both navigators and scientists, can be approximated by polynomials, so a difference engine can compute many useful sets of numbers.



The historical difficulty in producing error free tables by teams of mathematicians and human "computers" spurred Charles Babbage's desire to build a mechanism to automate the process.

The principle of a difference engine is Newton's method of divided differences. If the initial value of a polynomial (and of its finite differences) is calculated by some means for some value of X, the difference engine can calculate any number of nearby values, using the method generally known as the method of finite differences. For example, consider the quadratic

polynomial with the goal of tabulating the values p(0), p(1), p(2), p(3), p(4), and so forth. The table below is constructed as follows: the second column contains the values of the polynomial, the third column contains the differences of the two left neighbors in the second column, and the fourth column contains the differences of the two neighbors in the third column:

x	$p(x) = 2x^2 - 3x + 2$	diff1(x) = (p(x+1) - p(x))	diff2(x) = (diff1(x+1) - diff1(x))
0	2	-1	4
1	1	3	4
2	4	7	4
3	11	11	
4	22		

The numbers in the third values-column are constant. In fact, by starting with any polynomial of degree *n*, the column number *n* + 1 will always be constant. This is the crucial fact behind the success of the method.

The initial values of columns can be calculated by first manually calculating ***N*** consecutive values of the function and by backtracking, i.e. calculating the required differences.

Your task is to develop a program that imitate Babbage's Difference Engine by calculating a polynomial that iterate up to 50, that is p(50), based on the first *n* iteration.

Input

The first line of the input is a single positive integer ***T*** indicating the number of test cases.

For each test case, the input format is as follows:

n p(0) p(1) p(2) ... p(*n*)

where *n* is the size of polynomial and $n < 6$.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the output that required integer in a single line, based on the first iteration.

Sample Input	Sample Output
3 2 2 1 4 2 0 6 14 3 1 2 9 28	Case #1: 4852 Case #2: 2750 Case #3: 125001

G Lights

Input	Standard Input
Output	Standard Output
Time Limit	3 seconds

Problem Description

A rectangular panel consists of ***R*** rows, with ***C*** bulbs in each row. The rows are numbered from 0 to ***R***-1, and the columns are numbered from 0 to ***C***-1. Some of the bulbs are turned on and some of them are turned off. The panel is connected to a board with switches. The switches are arranged into a rectangle with ***R*** rows and ***C*** columns (i.e., with the same dimensions as the panel).

Flipping each switch will cause some of the bulbs to change state: those that were turned on will become turned off, and vice versa. More precisely, if we flip the switch with coordinates (row,col), then all bulbs with coordinates (x,y) where x<=row and y<=col will change their states. In other words, flipping the switch at (row,col) switches all the bulbs in the rectangle with opposite corners (0,0) and (row,col).

You will be given a matrix containing the initial states of the bulbs. The character '1' represents a bulb that is turned on, and the character '0' represents a bulb that is turned off.

Calculate the minimum number of flips necessary to have all the bulbs on at the same time.

Input

First line of the input contains ***T*** the number of test cases. Each test case starts with a line containing 2 integers ***R*** and ***C***. Each of the next ***R*** lines contains ***C*** character each. These ***R*C*** matrix contains the initial state of the bulb. ***R*** and ***C*** is between 1 and 500 inclusive.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the minimum number of flips necessary to have all the bulbs on at the same time.

Sample input and output is on the next page.

Sample Input	Sample Output
6 3 7 0001111 0001111 1111111 3 7 0000000 0000000 0000000 3 7 1111111 1111111 1111111 1 5 01001 1 10 1010101010 4 4 0101 1010 0101 1010	Case #1: 1 Case #2: 1 Case #3: 0 Case #4: 3 Case #5: 10 Case #6: 7

H Rooks

Input	Standard Input
Output	Standard Output
Time Limit	4 seconds

Problem Description

You have a chessboard with ***N*** rows and ***N*** columns. Initially you have ***N*** rooks placed in various squares of the chessboard. Two rooks attack each other if they are on the same cell or in the same column or in the same row. In each step you can move a rook one cell up/down/left/right. You cannot move it diagonally. You have to move these ***N*** rooks into ***N*** different squares with minimum steps so that none of them attack each other.

Input

The first line of the input gives an integer ***T***, which is the number of test cases. Each test case starts with ***N***. Each of the next line contains 2 integers (between 1 and ***N*** inclusive) in each line denoting the row and column of the rook. ***N*** can be as big as 20000.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the minimum number of steps needed to move the rooks.

Sample Input	Sample Output
2 3 1 1 1 1 1 1 4 1 1 1 1 1 3 3 1	Case #1: 6 Case #2: 8

I Fibonacci

Input	Standard Input
Output	Standard Output
Time Limit	4 seconds

Problem Description

Depicted below is the Fibonacci sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

As you can see, each value from 2 onwards is the sum of the previous two values. Given ***P***, ***Q*** and ***n***, find the ***n***th fibonacci number modulus ***Q***.

Input

The first line of the input gives an integer T, which is the number of test cases. Each test case consists of two integers ***P*** and ***Q***. ***P*** is between 1 and 2,000,000,000 inclusive. ***Q*** is between 1 and 10,000 inclusive.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the ***P***'th fibonacci number modulus ***Q***.

Sample Input	Sample Output
10 5 10 6 25 10 21 32 43 100 100 50 50 25 25 45 67 109 32 128 128	Case #1: 5 Case #2: 8 Case #3: 13 Case #4: 15 Case #5: 75 Case #6: 25 Case #7: 0 Case #8: 19 Case #9: 9 Case #10: 69

J

Prefix Free Subsets

Input	Standard Input
Output	Standard Output
Time Limit	3 seconds

Problem Description

A prefix-free set is a set of words in which no element is a prefix of another element in the set. For example {"hello"} , {"hello", "goodbye", "giant", "hi"} and the empty set are examples of prefix-free sets. On the other hand, {"hello","hell"} and {"great","gig","g"} are not prefix-free. You will be given a set of words, and you must calculate the number of subsets of words that are prefix-free. Note that both the empty set and the entire set count as subsets.

Input

The first line of the input gives an integer ***T***, which is the number of test cases. Each test case starts with a line containing ***N*** the number of words in the set. ***N*** is between 1 and 62 inclusive. Each of the next ***N*** line contains a word. The word contains only lowercase letter and length will not exceed 100.

Output

For each test case, the output contains a line in the format Case #x: ***M***, where x is the case number (starting from 1) and ***M*** is the number of prefix free subsets.

Sample input and output is on the next page.

Sample Input	Sample Output
3 3 hello hell hi 4 a b c d 6 a ab abc abcd abcde abcdef	Case #1: 6 Case #2: 16 Case #3: 7

--- End of problems ---