A Partition

Input: Standard Input
Output: Standard Output

The problem description is short and simple!
You will be given coordinates of **N** points in the 2D plane. You have to find the number of pairs of points, such that, if you draw a straight line through that pair, two conditions are satisfied:

- The straight line contains only two points, i.e. it only contains that pair.
- Each side of the straight line contains equal number of points.

Note that, if there are multiple points with the same co-ordinate, you have to consider those as one. To be more specific, you have to delete duplicate points.

Input

The first line will contain **T**, the number of test cases.
Of each test cases:
The first line will contain **N**, the number of points. Following **N** lines will contain two space separated integers denoting the coordinates of **N** points.

- **T ≤ 100**,
- **0 < N ≤ 1000**,
- Absolute value of coordinates **≤ 100000**.

Output

Output the number of desired pairs in a single line. Check the samples for more clarity.

Sample Input	Sample Output
2 6 0 0 0 1 1 0 1 1 1 1 1 1 6 0 0 0 1 1 0 1 1 2 2 3 3	Case 1: 2 Case 2: 2

B Graph Theory

Input: Standard Input
Output: Standard Output

In Computer Science, graph theory is a very important topic to learn. As we are already engaged with competitive programming alongside our studies, we all know different kinds of terms of graph theory like what is vertex, edge, cyclic or acyclic graph and so on. Recently Mr. Foo came up with two new terms of graph theory which are new to him. Let me share those things with you too:

- **Edge disjoint path:** Any two paths in a graph will be called edge disjoint path if their edge sets are disjoint
- **Vertex disjoint path:** Any two paths in a graph will be called vertex disjoint if their sets of **Inner vertices** are disjoint.
    - **Inner vertices:** A node **X** situated on the path from node **U** to **V** will be called inner vertex if **X ≠ U && X ≠ V**. i.e: If a path from **U** to **V** is: **U → A → B → C → V** then **A**, **B** & **C** will be inner vertices.

After knowing about these things, Mr. Foo started solving problems on these topics. Now he can easily solve problems where he needs to take care of **"edge disjoint path"** or **"vertex disjoint path"** separately. But recently he faced a problem where he needs to handle both of these things together! Again, to make his life even worse, the problem contains multiple queries too!!! In this situation, he has come towards your team to help him out of this situation!

The problem says that you will be given a directed acyclic graph (DAG) of **N** nodes rooted at node **1**. Now you will have to answer **Q** queries. The i-th query will contain two integers U_i and V_i which actually denote two nodes from the DAG. You have to say if you can find two paths: **1** to U_i and **1** to V_i such that these two paths are both edge disjoint and vertex disjoint at the same time. There can be multiple paths for **1** to U_i but you will have to choose any one from those. Same goes for the path from **1** to V_i. But these two chosen paths should be both edge disjoint and vertex disjoint to each other.

Input

The first line of the input will contain an integer **T(1 ≤ T ≤ 10)**.
In each test case, the first line will contain two integers **N (1 ≤ N ≤ 100000)** and **M (N-1 ≤ M ≤ 100000)** denoting the number of nodes and the number of edges in the in the directed acyclic graph respectively. Each of the next **M** lines will contain two integers U_i and V_i **($1 \le U_i, V_i \le N$ and $U_i \ne V_i$)** which means there is a directed edge from U_i to V_i. It is guaranteed that there will be no duplicate edges.
After the inputs, there will be an integer **Q (1 ≤ Q ≤ 100000)** denoting the number of queries in the problem. Each of the next **Q** lines will contain two integers U_i and V_i denoting two nodes from the DAG.

Output

For each test case, the first line should contain the case number in the format: "Case X:" where **X** is the number of test cases. Each of the next **Q** should contain the answer of the queries - "YES" if you can find two paths which are both edge and vertex disjoint otherwise "NO".

Birthday Chocolate

Sample Input	Sample Output
1 7 7 1 2 2 3 2 4 2 6 3 5 4 5 1 7 2 4 7 3 6	Case 1: YES NO

C Birthday Chocolate

Input: Standard Input
Output: Standard Output

It's your birthday and you got a lot of chocolates from your friends. You want to share those chocolates with your cousins. You have **N** cousins. You asked your cousins to stand in a row to get chocolates. Your friends want to make the chocolate sharing more interesting. So they set some constraints for you.

1. Each person will get at least 1 chocolate
2. Number of chocolates given to each person will be distinct
3. The **i+1 th** person will get more chocolate than **i-th** person if he is taller than **i-th** person, otherwise, he will get less chocolate than **i-th** person

They asked you to get a sequence of number where **A** of length **N**, where **i-th** person will get **A[i]** chocolate. But there are a lot of sequences which ensures those constraints. So they asked for the lexicographically smallest sequence which satisfies the constraints.

Input

The first line of input will contain an integer **T** (number of test cases).
For every test case, there will be **2** lines of input. The first line contains **N**, denoting the number of cousins you have. And the next line contains a string **S**. The string contains only the letters **'t'** or **'s'**. **S[i] = t** means **i+1 th** cousin is taller than **i-th** cousin in the row. **S[i] = s** means **i+1 th** cousin is shorter than **i-th** cousin in the row.

1 ≤ T ≤ 100
5 ≤ N ≤ 100000
|S| = N-1

Output

For every test case print 2 lines. In the first line, print `**Case x:**` where **x** is the test case number. In the second line, print **N** distinct integers, denoting the number of chocolates given to each cousin, separated by single space character.

Sample Input	Sample Output
3 5 tttt 5 ttss 5 sstt	Case 1: 1 2 3 4 5 Case 2: 1 2 5 4 3 Case 3: 3 2 1 4 5 1 2 3 4 5

D Yet Another Subset Sum Problem

Input: Standard Input
Output: Standard Output

You are given an **N x N** grid, where each cell in the grid is either blocked or free. You can place any number between **1** to **N** in the free cells. You are required to assign numbers to the free cells. No free cell should be left empty after the assignment, i.e. each free cell must contain **exactly** one number from the range **[1, N]**.

You are also given a target value **V**. Find out how many ways you can assign numbers to the free cells such that their sum equals **V**. It is also required to ensure just one more constraint, **no two rows or columns can contain the same number.**

Two ways are considered different if **any** particular cell has different values assigned.

Let **M** denote the total number of free cells**. It is guaranteed that there will be at least 1 and no more than 12 free cells.**

Input

The first line contains an integer **T**, denoting the number of test cases. The next line contains two integers, **N** and **V**. The next **N** lines contains the grid, each line contains the corresponding row of the grid.

Cells marked with a **.** indicates free cells, and cells marked as **X** denotes blocked cells. Each cell can be either blocked or free so the grid will not contain any other extra characters.

Constraints

$1 \leq T \leq 120$
$1 \leq N \leq 12$
$1 \leq V \leq 144$
$1 \leq M \leq 12$

Notes

For 60% of the test cases, the following constraint will hold:
$1 \leq N \leq 8$
$1 \leq M \leq 8$

Output

For each test case, output the case number followed by the number of ways to create the assignment.

J

Good Friends

Input: Standard Input
Output: Standard Output

World of Warships is a massively multiplayer online (MMO) game where you take your historical warships to the high seas again and fight as a team of 12 players against an enemy team of 12 other players. Now, as in any other MMO, there are good and bad players. Good players in the game do not want very bad players to be in their team because they make it hard to win the battles, similarly, bad players find it very hard to play against good players. So in a Reddit discussion thread, a keyboard war was waged among players asking game developers to distribute players among leagues, where players from the same league are more or less similarly skilled.

When developers started to rank **N** players in order to distribute them among leagues, the relationship among players sharply started to turn sour. The bottom ranked players became jealous of top ranked players and the top ranked players started to look down upon the bottom ranked players. They quickly started making friends among themselves who were closely ranked from each other.

Two players became friends when their ranks differed by at most **K** positions. For example, if **K = 1**, then players right next to each other in the ranking became friends. Furthermore, two players became good friends if they were friends and their IGN (in-game-names) had the same length. Write a program to find out the number of pairs of good friends in a given rank list. Players are ranked from highest to lowest ranks from top to bottom respectively in the given rank list.

Input

The first line of input contains an integer **T** (**1 ≤ T ≤ 10**), the number of test cases. Each of the test cases starts with two positive integers, **N** (**3 ≤ N ≤ 300000**) and **K** (**1 ≤ K ≤ N**), as described in the description above, then the following **N** lines contains a single string which is the player's IGN consisting of 2 to 20 uppercase English letters.

Output

For each case, print a line containing the number of pairs of good friends as described above.

Sample Input	Sample Output
1 4 2 IVA IVO ANA TOM	5

K

This is the Giveaway

Input: Standard Input
Output: Standard Output

Usually, the easiest problem in the contest is called giveaway. Because the judges feel, that we are basically giving it away to the teams. And young judges are terrible at setting giveaway problems. Because the problem they think is easy can often turn out to be NOT SO easy. That's why the oldest person in the judging panel sets the giveaway.

While trying to set the giveaway for this contest, I was desperate to avoid any string related problems. Finally I could come up with a problem that looks like a string problem but actually is not. So, in this problem you have to print just one line: "This is the giveaway". But sometimes a character of this line gets missing and that's denoted using an integer **K**. So if **K = 1**, the first (leftmost) character is missing, and then the line printed will look like "his is the giveaway". If **K = 5**, then printed line will look like "Thisis the giveaway". If **K = 0**, you may assume that no character is missing. Also if **K** is larger than the length of the string, that also means that no character is missing.

Given **K** as input, you need to print "This is the giveaway" in the desired way as described above.

At first glance, the problem may seem impossible to solve without string. But you need to consider the fact that, the string is fixed. And so it can be solved only using "if else". The code will get complex in that way, but you can't deny that "if else" is an absolutely valid way to solve this problem.

Input

The first line will contain an integer **T** (**0 < T < 25**), the number of test cases. **T** lines will follow. Each line will describe a case. Each case will have only one input, an integer **K** (**0 ≤ K < 25**).

Output

For each case, print the desired line. See sample for clarification.

Sample Input	Sample Output
4 1 2 0 8	his is the giveaway Tis is the giveaway This is the giveaway This isthe giveaway

E Tribonacci

Input: Standard Input
Output: Standard Output

The tribonacci numbers are like the fibonacci numbers, but instead of starting with two predetermined terms, the sequence starts with three predetermined terms and each term afterward is the sum of the preceding three terms. If the first three numbers are **1**, **2** and **3** then the first few tribonacci numbers are:

1, 2, 3, 6, 11, 20, 37, 68, ...

You are given a number **N**. You have to calculate the **N**th tribonacci number. If the number has more than **K** digits, print the last **K** digits of the number.

Input

Input starts with test case number **T** ($1 \le T \le 5x10^5$). For each of the test case, each line contains two space separated integers **N** ($1 \le N \le 10^{18}$) and **K** ($1 \le K \le 6$).

Output

For each case of input, output the answer of the problem in the format "Case X: Y" where **X** denotes the case number,and **Y** denotes the answer. See the sample input output for more understanding.

Sample Input	Sample Output
3 4 2 7 1 7 3	Case 1: 6 Case 2: 7 Case 3: 37

N.B. Dataset is large. Use faster I/O methods.

F Expected Coin Change

Input: Standard Input
Output: Standard Output

Mr. Kiptus has a bag full of infinite number of coins of value **1** to **N** Taka (he is rich). He wants to donate some coins. He has decided to not give more than **M** Taka (he is also kiptus). But he is also interested to keep things dynamic. So he devises the following procedure:

1. He will first start with **0** donations.
2. He will take a coin randomly from the bag. The probability of getting any coin from **1** to **N** is same.
3. He will then check if on donating this coin, his total donation exceeds **M** or not. If not, then he will donate the coin and start again from step 2, and his donation will increase by the coin's value.
4. Otherwise, he will return the coin to the bag and stop donating.

Now he is wondering what is the expected amount he will donate. Can you calculate it?

Input

The first line of the input is **T** (**T ≤ 2000**), then **T** test cases follow. In each case, there will be two integers **N** (**2 ≤ N ≤ 2000**) and **M** (**0 ≤ M ≤ 10000000**).

Output

For each test case, print a line of the format "Case I: F", where **I** is the case number and **F** is the expected money Mr. Kiptus will donate. Absolute error upto **1E-4** will be ignored.

Sample Input	Sample Output
5 3 5 5 3 10 100 100 10000000 1000 10000000	Case 1: 4.300412 Case 2: 1.368000 Case 3: 97.000000 Case 4: 9999967.000000 Case 5: 9999667.000000

G

Forex Man

Input: Standard Input
Output: Standard Output

One day Tom suddenly developed a superpower. He got the ability to see how the value of USD currency fluctuates against BDT currency in the next **N** days of future. I know, that's a very specific superpower to develop, but that's what he got. It's not like I am making up the story.

Why am I saying it's an incredible superpower? Because using the superpower one can make a lot of money. If one buys a currency when it has low value and then sells that currency when they have high value, that means profit! And unlike normal share market, where you have to buy before selling, Forex (foreign exchange) market allows you to sell before you buy! That means you can even sell a currency when it has high value and buy it back when it has low value, allowing you to make money in both ways.

Confused? Well, I suppose if you don't have knowledge about Forex then it will confusing. Well, guess what, Tom was confused too! So created a simplified model which was easier to understand for programmers.

So Tom modeled the prediction of currency value for the next **N** days as an array of **N** integers. Here each ith integer denoted the predicted value of the currency on the ith day.

Now, in order to not attract unwanted attention from secret government agencies that deal superpower containment, Tom decided to make a limited profit. So Tom decides on a value **K** and he will be happy to use his powers as long as he makes at least **K** profit.

Now, he wants to select two days from the next **N** days (i.e, select two integers from the array, let them be **x** and **y**). Let the first day selected be **A** and second day selected be **B**. He then proceeds to sell the currency on day **A** and buy the currency in day **B**. Since it's forex, he can sell the currency before he buys it, no big deal.

After the transaction, he makes a hefty profit which equals to profit = sell-buy. Now Tom has some conditions:

- The value of the currency on day **A** should be higher than the value of currency on day **B** (since he is selling on day **A** and buying on day **B**).
- Profit must be greater than or equal to **K**.
- If multiple selections of days result in profit ≥ **K**, then transaction period should be as small as possible, i.e, **|A - B|** should be as small as possible, where **A** and **B** are the positions of the elements in the array.
- If there are still multiple solutions, select the one with smallest **A**.
- If there are still multiple solutions, select the one with the biggest profit.
- If there are still multiple solutions, select the one with smallest **B**.

For example, if **N = 7**, the model array is arr [**1,1, 4, 5, 9, -1, 3**] and **K = 4**, some possible answers are:

- A=3, B=0, arr[3]-arr[0] = 5 - 1 = 4
- A=4, B=3, arr[4]-arr[3] = 9 - 5 = 4
- A=4, B=2, arr[4]-arr[2] = 9 - 4 = 5
- A=4, B=0, arr[4]-arr[0] = 9 - 1 = 8
- A=6, B=5, arr[6]-arr[5] = 3 - (-1) = 4
- A=4, B=5, arr[4]-arr[5] = 9 - (-1) = 10

Out of the **5** possible answers, **{A,B} = {4,3}**, **{6,5}** and **{4,5}** have the smallest range. Out of these three, **{4,3}** and **{4,5}** has smallest **A**. Between these two, **{4,5}** has bigger profit, so we select **{4,5}** as our answer.

Input

The first line of the input will contain a single integer **T** (**T ≤ 10**), denoting the number of the test case. Next specifications of **T** test cases will follow.

Each test case starts with a single integer **N** (**2 ≤ N ≤ 10000**) denoting the number of elements in the array A. Next line will contain **N** integers separated by whitespace, where each number will be a signed 32-bit integer. These **N** integers from the array **A** as mentioned above.

Next line will contain a single integer **Q** (**1 ≤ Q ≤ 100**) denoting the number of queries. Next will contain **Q** integers denoting the value of **K**. The values of **K** will fit into a signed 32-bit integer.

Output

First, print the test case number and then for each query of a test case, print the value of **A** and **B** in a single line. It is guaranteed that an answer will always exist. See sample input/output for more details.

Sample Input	Sample Output
1 7 1 1 4 5 9 -1 3 1 4	Case 1: 4 5

H

Moving Balls

Input: Standard Input
Output: Standard Output

"Alice and Bob are fictional characters commonly used in game theory, cryptography and in different fields of computer science. Ron Rivest, Adi Shamir, and Leonard Adleman introduced us with Alice and Bob in 1978. "

Alice and Bob started to play again. Initially, they have **N** piles of balls and these piles are numbered from **1** to **N**. K^{th} pile has H_K balls. If Bob puts a ball on the top to the K^{th} piles, the ball will move using the following recursive function:

```
Move (K)
    {
    if (K==1) Stop.
    if (H_{k-1}≤H_k ) Move (K-1);
    Else Stop.
    }
```

The game has **Q** operations. During each operation Bob has to perform any of the followings:

- Put **Y** balls one by one on the top of X^{th} Pile.
- Determine **Sum** = $\sum_{K=X}^{Y} H_K$

Bob never wants to lose a game. So, he wants your help.

Input

Input starts with an integer **T(1 ≤ T ≤ 10)** denoting the number of test cases. Each of the test cases starts with two integer **N(1 ≤ N ≤ 100000)** and **Q(1 ≤ Q ≤ 30000)**.
Next lines contain **N** space separated integers $H_1, H_2, \ldots H_n$ **(1 ≤** H_K **≤ 100000)** denoting the number of balls on each pile.
Next **Q** lines contain the operations in the following form:

1 Y X: Put **Y** balls one by one on the top of X^{th} Pile where **1 ≤ Y ≤ 100000** and **1 ≤ X ≤ N**.

2 X Y: Determine **Sum** = $\sum_{K=X}^{Y} H_K$ where, **1 ≤ X ≤ Y ≤ N**.

Output

For each test case, the first line of the output should contain the case number in the format: **"Case T:"**, where T is the test case number. For each of the operation of **type = 2**, print the desired answer in a single line.
For more clarity, please see the sample I/O.

Sample Input	Sample Output
1 6 10 2 5 9 6 10 8 1 2 3 2 1 1 2 1 3 1 2 4 2 1 3 2 1 4 1 5 6 1 8 1 1 5 2 2 2 5	Case 1: 4 18 18 26 38

Explanation

	Operation Details	Number of balls on each pile
		2 5 9 6 10 8 (Initially)
Q=1	Put 2 balls on 3rd Pile one by one	4 5 9 6 10 8
Q=4	Put 2 balls on 4th Pile one by one	4 5 9 8 10 8
Q=7	Put 5 balls on 6th pile one by one	6 5 9 9 10 10
Q=8	Put 8 balls on 1st pile one by one	14 5 9 9 10 10
Q=9	Put 5 balls on 2nd pile one by one	14 10 9 9 10 10

NB: Data set is large. Use fast I/O methods.

I Help in maintaining communities

Input: Standard Input
Output: Standard Output

There are **N** houses located in a row from left to right. 1st house is located at index 1 and N^{th} house is located at index **N**. Each house has its **type** and **friendliness** value in the society.

A **community** is defined by the group of all houses from L^{th} index to R^{th} index (inclusive, **L ≤ R**), where the houses at L^{th} and R^{th} index **should have the same type.** Each house should be **exactly** in one community.

A **single house** is also considered as a community, in that case, the community **should pay security tax** of that house to the government.

The cost to maintain a single community (houses from **L** to **R**) is called **maintenance cost** of that community denoted by **MC,** which is calculated as:

- If the community consists of a single house then **MC = security tax of that house**.
- **Otherwise,**

$$MC = \sum_{i=L}^{R-1}((friendliness[i] - friendliness[R])^2 - friendliness[R]^2) * IsSame(type[i], type[R])$$

- Where:
  - IsSame(**x**, **y**) = If **x** and **y** are same then **1** else **0**.

Total maintenance cost (**TMC**) of N houses is the summation of maintenance cost of all communities. For this problem, **TMC** can also be **negative**.

Your task is to divide the houses in any number of valid communities in any possible ways, which ensures the **TMC** is **minimum**.

Example:

Index	1	2	3
House type	1	2	2
Security tax	100	50	150
Friendliness value	10	8	9

Two possible arrangements are:

1. [{ 1st house }, { 2nd house, 3rd house }], here are two communities. **TMC = 100 + $(8-9)^2 - 9^2$ = 20**.
2. [{ 1st house }, { 2nd house }, { 3rd house }], here are three communities. **TMC = 100 + 50 + 150 = 300**.

Input

The first line contains an integer **T**, denoting the number of test cases.
For each test case:

- The first line contains an integer **N** denoting the number of houses.
- The second line contains an array of **N** space separated integers denoting the type of **N** houses.
- The third line contains an array of **N** space separated integers denoting security tax of **N** houses.
- The fourth line contains an array of **N** space separated integers denoting friendliness value of **N** houses.

Constraints:

- **1 ≤ T ≤ 5**
- **1 ≤ N**, Type of house **≤ 100000**
- **-100000 ≤** security tax, friendliness value **≤ 100000**

Output

For each test case, print the minimum **TMC** possible.

Sample Input	Sample Output
4 3 1 2 2 100 50 150 10 8 9 3 1 7 1 50 1500 50 1 8 1 3 2 5 2 4 1121 7 4 8 6 3 662 665 665 100000 100000 100000 -100000 -100000 -100000	20 -1 -32 -9999900000

N.B. Dataset is large. Use faster I/O methods.