The 2016 ACM ASIA Region Programming Contest

Kolkata Site

Onsite Contest Problems

Sponsored by IBM

Organized by NITTTR Kolkata and JIS Group

Problem code: KOL16A
Problem name: Playing World War II

Enigma: The military machine used by Nazi Germany to encipher messages before and during World War II. The code of Enigma was broken once by Polish Cipher Bureau in 1932 and then again, during the war, by the Codebreaker team at Bletchley Park, England. Because of this, the Allied Powers were able to intercept a lot of messages transferred by German military and knew their battle plans. It is said that, breaking of Enigma shortened the length of that war by two years.

But, in the time of war, Allies were gravely concerned about the possibility of Germans finding out this security breach and resetting their communication system. They were very disciplined about using the information in order to hide this source of intelligence. Their precautions included leading fake invasions, superfluous search missions and attacking only after creating a proper cover story. Even at times, they considered sabotaging own resources during a battle which resulted in the death toll of hundreds of soldiers, civilians and massive loss on supplies; just to avoid any suspicion!

It seems like they played with probabilities leaving peoples' lives hanging on the decision.

Cruel, wasn't it? Can you do it?

There are **N** battles ahead. The probability that Allies will win the i^{th} battle is $\mathbf{P_i}$.

The enemies are calculating a value **PEB (Probability that Enigma has been broken)**. If the allies win a battle, the value of PEB increases; if the enemy wins the battle, the value decreases. If the value of PEB gets too high, the enemy will become suspicious and replace Enigma with something else.

PEB is calculated as an average from results of all previous battles using the following method.

```
total := 0
foreach Battle, b_i
        if Allies win
                total += probability of Allies losing in b_i
        else
                total -= probability of Allies winning in b_i
PEB = max(total / no. of battles, 0)
```

As the Allies already deciphered Enigma, they can choose to win or lose any battle they want. Consider the following scenario as example:

Battle No.	Probability, Allies Win	Probability, Allies lose (1 – winning probability)	Decision	PEB
1	0.7	0.3	Win	0.3
2	0.4	0.6	Win	(0.3+0.6)/2 = 0.45
3	0.8	0.2	Lose	(0.3+0.6-0.8)/3 = 0.033
4	0.3	0.7	Win	(0.3+0.6-0.8+0.7)/4 = 0.2
5	0.9	0.1	Lose	(0.3+0.6-0.8+0.7-0.9)/5 = -0.2 -> 0.0
6	0.6	0.4	Win	(0.3+0.6-0.8+0.7-0.9+0.4)/6 = 0.05

Some battles are more important than others; the i^{th} battle has an Impact Factor $\mathbf{F_i}$. As a war analyst, your job is to maximize the sum of Impact Factors of all the battles won. But the enemies must not know that Allies have deciphered Enigma, so you must keep PEB less than a certain **threshold, S**; throughout the whole war.

Input

- The first line of input contains an integer **T ($1 \le T \le 100$)** denoting the number of test cases. Each test case contains 4 lines:
- The first line contains one integer **N ($1 \le N \le 100$)** denoting the number of battles.
- The second line contains **N** space-separated real numbers $\mathbf{P_1,P_2,....,P_n}$ (probability of the Allies winning in i^{th} battle) $\mathbf{0.00 \le P_i \le 1.00}$.
- The third line contains **N** space-separated integers $\mathbf{F_1,F_2,...,F_n}$ (Impact factor of the i^{th} battle). Here $\mathbf{1 \le F_i \le 10000}$
- The fourth line contains a real number denoting threshold, **S ($0.00 \le S \le 1.00$)**.

The real numbers contain at most 2 digits after the decimal point.

Output

For each test case, print case number and the maximum sum of impact factors.

Sample

Input

```
2
2
0.8 0.9
100 200
1
4
0.28 0.62 0.92 0.96
3 8 7 2
0.05
```

Output

```
Case 1: 300
Case 2: 9
```

Problem code: KOL16B
Problem name: Extreme Encoding

Lajuk is a little girl who loves playing with array. In her 10th birthday, she got two arrays as presents. Let's call them **A** and **B**. Both arrays have the same size **n** and contains integers between **0** to **30000**.

Lajuk's hard-drive is almost full of presents and she barely has any space to keep the arrays. She discovered a brilliant function to merge the array into one:

```
int encodeInteger(int x, int n){
        n = n<<(1<<(1<<(1<<1)));
        x = x | n;
        return x;
}
void encodeArray(int *A, int *B, int n){
        for(int i=0;i<n;i++) {
            A[i] = encodeInteger(A[i], B[i]);
        }
}
```

Lajuk passed **A** and **B** into the encodeArray function. After that she discarded array **B** and saved the modified version of array **A** in the hard-drive.

Now in her 15th birthday Lajuk wants to play with those arrays again. She found the modified version of array A in the hard-drive but she forgot how to recover the original arrays from it. Being upset, she asked for your help. Can you help her to recover the original arrays?

Input

The first line contains **T ($1 \leq T \leq 100$)**, the number of test cases. The first line of each test cases contains **n ($1 \leq n \leq 10^4$)**, the size of the array. Next **n** line contains **n** integers denoting the modified array **A**.

Output

For each case print the case number in the first line. In the second line, print n integers denoting the original array **A**. In the third line print n integers denoting the array **B**. Two consecutive integers should be separated by a single space.

Sample

Input

```
1
4
196613
655370
196620
131079
```

Output

```
Case 1:
5 10 12 7
3 10 3 2
```

Problem code: KOL16C
Problem name:

Factorial n or n! is defined as below:

```
n! = 1*2*3*...*n
```

For example **7! = 1*2*3*4*5*6*7 = 5040**. Now, if we prime factorize a factorial we will be able to find the frequency of each prime in it For example **7!= 5040 = 2^4*3^2*5*7**. So there are exactly four **2**'s, two **3**'s, one **5** and one **7** in **7!**. For factorials of large numbers prime-factorization is not a good idea and there are many better ways to find out frequency of prime numbers it factorials of large number. Now consider **15** numbers **1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, 9!, 10!, 11!, 12!, 13!, 14!, 15!** and a set of consecutive prime numbers **{3, 5, 7, 11, 13}** (2nd, 3rd, 4th, 5th and 6th) and an integer **2** (The frequency) and you are asked to find how many of these numbers have one prime from the set of primes exactly two times. The answer will be **9** and these 9 numbers are shown below:

Value	Primes that are present exactly 2 times
6!	3
7!	3
8!	3
10!	5
11!	5
12!	5
13!	5
14!	5, 7
15!	7

In this problem you are asked to solve a generalized version of this problem. Given a set of factorials of consecutive numbers **F**, a set of consecutive prime numbers **P** and an integer **t**, you job is two find out how many of the numbers in **F** contains any one of the prime numbers from set **P** exactly **t** times. But if more than one prime from **P** is present exactly **t** times in some number, it is ok.

Input

The first line contains a positive integer **C (0<C<51)** which denotes the total number of test cases to follow. Each of the next **C** lines consists input for a single test case. Each line contains five positive integers **low, high (0 < low < high ≤ $2*10^{16}$), plow, phigh (0 < plow < phigh ≤ 3000000)** and **t (0 < t ≤ 300000)**. These numbers denotes that you have to deal with the set of factorial of consecutive numbers **low!, (low+1)!, (low+2)! ..., (high-1)!, high!** and a set of primes **P** which contains P_{plow}, P_{plow+1}, ..., P_{phigh}. Primes are numbered starting from **1**, so P_1 is actually **2,** P_2**=3,** P_3**=5,** P_4**=7,** P_5**=11** etc. You will have to find how many of these factorials contains one or more primes from this set **P** exactly **t** times.

Output

For each test case produce one line of output. This line contains at integer **M**, which denotes how many of the numbers **low!, (low+1)!, (low+2)! ..., (high-1)!, high!** contains at least one prime from set **P exactly t times.**

Sample

```
Input
2
10 20 1 5 3
10 20 1 5 6

Output
5
3
```

Problem code: KOL16D
Problem name: Men and Horses

There are $\mathbf{n_m}$ men living in a town. They have $\mathbf{n_h}$ horses, each capable of carrying only one man at a time. All of them are supposed to attend a dinner party **d** km away from their house. They should start their journey together and reach destination in minimum time. Speed of each man is $\mathbf{v_m}$ km/hour whereas speed of each horse (with or without somebody on back) is $\mathbf{v_h}$km/hour. You need to find the minimum time required to complete the journey? For simplicity you can assume the following things:

- No time is needed in riding a horse or getting down from a horse.
- The horses are very well trained to execute any instruction given to it.
- No time is needed for changing the direction of horse or man.
- The town and the dinner party location can be considered as two different points and all people and horses travel along the straight line connecting these two points.

Constraints

- d ($0 < d \leq 10000$)
- n_m ($1 < n_m \leq 1000$)
- n_h ($0 < n_h \leq 1000$)
- v_m ($0 < v_m \leq 20$)
- v_h ($0 < v_h \leq 50$)

Input

Each input set consists of five integers **d, n_m, n_h, v_m, v_h** in a new line. Input ends with five 0s in a line. There will be at max $\mathbf{10^5+1}$ lines in the input

Output

For each line of input except the last one produce one line of output. This line should print a fraction **p** by **q**, in **p/q** form. This denotes the minimum possible reaching time of all men. Here **p** and **q** must be relative prime.

Sample

Input

```
70 2 1 5 10
0 0 0 0 0
```

Output

```
49/5
```

Problem code: KOL16E
Problem name: Divide Me Please

There are some short-cut ways to check divisibility of numbers in base 10. Here are some examples:

1. Remember divisibility testing of 3 in base 10? It was simple, right? We need to add all the digits and then check if it is divisible by 3. Call this method **"1-sum"**.
2. In case of testing of 11, we need to add all digits by alternating their signs. For example, 1354379988104 = 11*123125453464 and (4-0+1-8+8-9+9-7+3-4+5-3+1) = 0, which is divisible by 11 (0 = 0*11). Lets call this method **"1-altersum"**.
3. In case of 7, we need to add all 3-digit-groups by alternating their signs. For example, **8**618**727**529**993** = 7*1231246789999 and (**993**-529+**727**-618+**8**) = 581, which is divisible by 7 (581 = 7*83). Similarly, we call this method **"3-altersum"**.
4. In similar Manner, 13's checking is **"3-altersum"**.

In this problem, we are interested to see divisibility checking of **only** prime numbers in base 10. For a prime **P**, you need to find the smallest positive integer **N** such that **P's divisibility testing** is **"N-sum"** or **"N-altersum"**.

Input

At first you will be given **T (T ≤ 1000)**, the number of test cases. Then **T** lines will follow. In each line there will be single integer **P (P ≤ 2^{31}-1 = 2147483647)**, the prime number (**P** is a prime number **NOT** equal to **2** or **5**).

Output

For each line of input, produce exactly one line of output like either "Case <test-case>: <N>-sum" or "Case <test-case>: <N>-altersum". Please see sample input and output for details.

Sample

Input

```
4
3
7
11
13
```

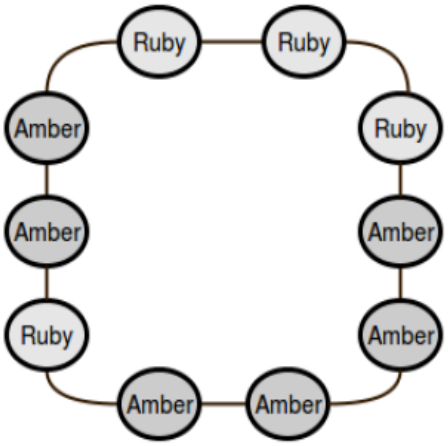
Output

```
Case 1: 1-sum
Case 2: 3-altersum
Case 3: 1-altersum
Case 4: 3-altersum
```

Problem code: KOL16F
Problem name: Precious Stones

Nemo went to the shop to buy a chain for his girlfriend. The shopkeeper has shown him a chain made of some expensive stones. The chain has **n** stones marked from **0** to **n-1**. The **i^{th}** stone is connected with **$((i+1)\%n)^{th}$** stone for each **$0 \leq i < n$**. Each stone can be either Ruby or Amber.

Nemo defines beauty factor **B** of a chain as the maximum number of consecutive stones of same type.



In the chain above, the beauty factor is **B = 4** because there are **4** consecutive Amber in the chain.

Nemo wants to chose **exactly one** stone **i** and exchange it with a stone with different type. So if the **i^{th}** stone is ruby, Nemo will exchange it with Amber and vice-versa. He wants to do it in such a way that the value **B** is as small as possible.

Given the configuration of the chain, can you find the minimum value of **B** that Nemo can get after exchanging **exactly one** stone? Note that, It's not allowed to change positions of the stones, the new stone must be placed in the same position as the original stone.

Input

First line contains number of test cases **T (1 ≤ T ≤ 2500)**. For each test cases, there is a single line containing a string **S (1 ≤ |S| ≤ 10^5)** denoting the chain. Here |S| denotes the length or number of characters in the string. The string is made of only **'R'** (Ruby) and **'A'** (Amber). Total number of characters in the input file will be less than **5 x 10^6**.

Output

For each test case, print the case number and the answer in a single line. Look at the output for sample input for details.

Sample

Input

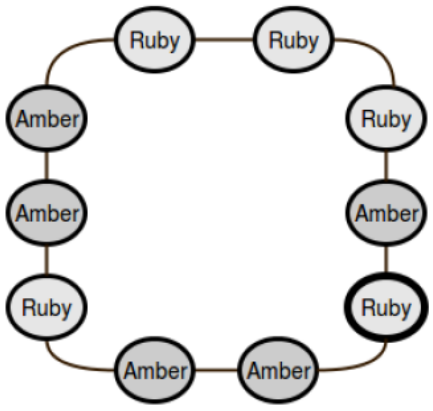
```
2
RRRAAAARAA
ARRRRAAA
```

Output

```
Case 1: 3
Case 2: 4
```

Explanation:

Case **1** represents the figure in the problem statement. Nemo can exchange one Amber with a Ruby and reduce **B** to **3**.



Problem code: KOL16G
Problem name: Optimal use of Nuclear Power plant

It is now 2050 AD. A city will be built under large protective circular dome. A nuclear power plant **(NPP)** built outside the city will power the whole city. For safety reasons the **NPP** must be at least $\mathbf{d_{min}}$ distance away from the dome. The radius of the dome has not been fixed yet. The houses inside the dome will be built after building the dome. So for simplicity it can be assumed that each location under the dome is equally likely to have a house. In other words houses will be uniformly distributed in the land inside the dome. As the houses are full of modern electrical equipments, so the power consumption per house is quite high. So each house is connected with the **NPP** with the shortest possible cable underground. In the figure below you can see four houses (Blue circles) and their shortest connection with **NPP** (dotted line). For wiring cost and line-loss issues the average distance of the houses (assuming they are uniformly distributed under then dome) from the **NPP** cannot be more than $\mathbf{D_{avg}}$.

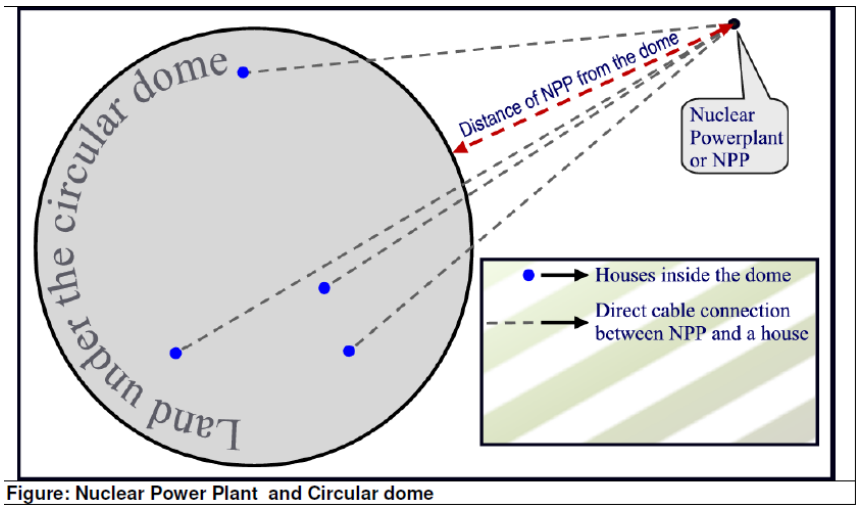


Figure: Nuclear Power Plant and Circular dome

Given the value of $\mathbf{D_{avg}}$, $\mathbf{d_{min}}$ your job is to find the maximum possible value of the radius of the land under dome.

Input

First line of the input file contains an integer **T ($0 < T \leq 20$)** which denotes how many test cases to follow. Each of the next **T** lines contains two strictly positive integers $\mathbf{D_{avg}}$ **($0 < D_{avg} < 1000$)** and $\mathbf{d_{min}}$ **($0 < d_{min} < D_{avg}$)**. The meaning of $\mathbf{D_{avg}}$ and $\mathbf{d_{min}}$ are given in the problem statement above.

Output

For each test case produce one line of output. This line contains a floating-point number $\mathbf{R_{max}}$, which denotes maximum possible value of the radius of the circular land under the dome maintaining the constraints imposed by $\mathbf{D_{avg}}$ and $\mathbf{d_{min}}$. This value should be rounded to four digits after the decimal point. Errors not exceeding $\mathbf{10^{-4}}$ will be ignored.

Sample

Input

```
2
100 20
10 5
```

Output

```
72.6679
4.7114
```

Problem code: KOL16H
Problem name: Mr Wireless

Mr. Wire Less is not that good at implementing circuit in a breadboard. In his Digital Logic Design course, he has to implement several boolean functions using the breadboard. In a breadboard, inputs are given through the switches and outputs are taken through the LEDs. Each input switch can be either in ground state or in high state. So, if he wishes to implement a boolean function, $\mathbf{f(x_1, x_2, .., x_n)}$ that takes **n** boolean values as input and returns **m** boolean values as output, he will use **n** switches and **m** LEDs.

Mr. Wire Less can quickly assemble the necessary ICs and wires, but the key problem is testing. While testing he has to check with all possible input combination, to make sure whether the output of LED corresponds to the expected output or not. This is taking too long for him, as most of the switches are jammed and difficult to toggle.

Mr. Wire Less is asking for help to minimize his time of testing. So, your task is to minimize the total number of switch-toggle throughout the testing.

For example, if Mr. Wire Less has to test a function **f(x0, x1)** of two variables, he may choose this switching-sequence for testing **00, 11, 10, 01**. In this case, the total number of switch-toggle will be **2+1+2 = 5**. But if he tests in this sequence **00, 10, 11, 01** total number of toggle will be **1+1+1 = 3**.

Given **n**, you have to output the minimum number of toggle needed for complete testing. Though it seems impractical, he wants you to solve the problem for a very large value of **n**. But, then the toggle value can be quite big. So, he is completely okay with the toggle value modulo **8589934592**.

Input

The first line of the input contains a positive integer **T($T \leq 10^5$)**, denoting the number of test-case. Each of the following **T** lines contains a single non-negative integer **n($n \leq 10^{20}$)**.

Output

For every test-case, output a single containing test-case number and the minimum number of switch-toggle modulo **8589934592**.

Sample

Input

```
2
1
2
```

Output

```
Case 1: 1
Case 2: 3
```

Problem code: KOL16I
Problem name: Roads

Sarah is an engineer assigned to build roads between the **N** towns of Evoland. Towns are assigned IDs from **1** to **N**. A road connects exactly one pair of towns, and is bidirectional.

The mayor of Evoland allocated a very small budget for this project. Thus, Sarah is forced to build *as few roads as possible* while still keeping all **N** towns connected via roads. Note that this is an **absolute requirement**; any network not satisfying this will not be accepted by the mayor.

On top of this absolute requirement, Sarah still can't just build the roads however she wants. She has to consider the visiting habits of tourists, who are very picky with which towns they visit. She found out that tourists don't visit towns which are "too similar".

For example, consider **N = 4** towns. Suppose Sarah builds three roads connecting the pairs of towns **(1, 2)**, **(2, 3)** and **(2, 4)**. Then as you can see, town **2** will be the busiest town because all towns are adjacent to it by a single road. On the other hand, from the point of view of the roads, towns **1**, **3** and **4** will be pretty much the same, so tourists will tend to visit only one of them.

Sarah has formalized this notion of "sameness" of towns. For a given network of roads, she defines two towns **a** and **b** to be **indistinguishable** if there exists a permutation of the towns $[p_1, p_2, p_3, ..., p_N]$ such that $p_a = b$, and for every pair of nodes **(i, j)**, there is a road between **(i, j)** if and only if there is a road between (p_i, p_j).

On the other hand, Sarah calls a pair of towns **distinguishable** if they are not indistinguishable. (Unsurprisingly.)

She also defined the **attractiveness** of the network of roads as the size of the largest set of towns that are pairwise distinguishable from each other. For example, the network above has an attractiveness of **2**, because there are at most two distinguishable towns.

Sarah's goal is to build the roads in such a way that the attractiveness becomes exactly **A**. Can you help her achieve this by coming up with the road plan?

Input

The first line of input contains an integer **T** denoting the number of test cases. The description of **T** test cases follows.

Each test case consists of a single line containing two integers **N** and **A** separated by a space.

Output

For each test case, first output **R** in a single line. **R** denotes the number of roads to build. Then, output **R** lines, each containing two integers **i** and **j**, meaning that a road exists between **i** and **j**. There can be multiple possible correct outputs; any one will be accepted.

If it's not possible to make such a network of roads at all, just output **-1** in a single line.

Constraints

- **1 ≤ T, A, N ≤ 300**

Example

Input:

```
2
4 2
5 234
```

Output:

```
3
1 2
2 3
2 4
-1
```

Explanation

The first case corresponds to the network described in the problem statement. In the second case, it is impossible to find a network for **5** towns such that the attractiveness is equal to **234**, so we output **-1**.

Problem code: KOL16J
Problem name: Quentin Tarantino

Quentin Tarantino is a famous Hollywood filmmaker and actor. His films have a unique characteristic of connecting with youth by using popular culture references. They are usually divided into various sub-parts denoted by chapters. His plot of stories is never linear, he always make sure that the chapters should never be shown in the order of Chapter 1, Chapter 2, Chapter 3, ,.. Chapter **n** linearly. He believes doing this an insult to the intelligence of viewers. For example, Chapter 1, Chapter 2, Chapter 3 can never be a chapter sequence in Tarantino's movies.. He can have a sequence of Chapter 2, Chapter 3, Chapter 1 instead.

Recently Santa gifted Tom a movie which consists of **n** chapters. This movie can possibly be directed by your favorite Tarantino or by someone else. Directors in hollywood can sometimes be very sloppy, they sometimes drop some chapters too and stupid they are that they will leave numbering of the chapters as it is and one can easily identify the chapters that were dropped. For example, if you find the sequence of chapters as - chapter 1, chapter 5, chapter 3, don't be surprised, they did not even take burden of renumbering the chapters to chapter 1, chapter 3, chapter 2. They can sometimes repeat the chapter numbers too. Remember that Tarantino is not a sloppy director, so he never makes this kind of stupid mistakes.

Tom happened to find the sequence of chapters in the movie, this sequence contains **n** chapters, **i**-th of which is denoted by $\mathbf{chapter_i}$. Tom is a big fan of movies of Quentin. So, Tom would like to check whether the movie could possibly be directed by Tarantino or not? Please help Tom, he really needs it.

Input

The first line contains an integer **T** denoting the number of test cases. **T** test cases follow.

The first line of each test case contains an integer **n** denoting number of chapters of the movie.

The second line of each test case contains **n** space separated integers $\mathbf{chapter_i}$, denoting numbering of $\mathbf{i^{th}}$ chapter.

Output

For each test case, output "yes" or "no" in a separate line.

Constraints

- $1 \leq T \leq 100$
- $2 \leq N \leq 100$
- $1 \leq chapter_i \leq 500$

Example

Input:

```
3
3
2 3 1
3
1 5 3
3
1 2 3
```

Output:

```
yes
no
no
```

Explanation

All the examples are explained in the problem statement.

Problem code: KOL16K
Problem name: Chef and Points

Today, you finally have a chance to compete with a famous coder like Chef and prove your mettle.

Chef has **n** distinct points in 2-D plane, i.e. there are never two points with both **x** and **y** coordinate equal and there are no two points with same **x** coordinate. Consider a subset of above **n** points which can be arranged as a sequence $\mathbf{p_1},..\mathbf{p_k}$, such that $\mathbf{x_{p1}} < \mathbf{x_{p2}} < ... < \mathbf{x_{pk}}$ and each point $\mathbf{p_i}$, where **i** is in range between 3 and **k** (both inclusive) lies below or on the line passing between points $\mathbf{p_{i-1}}$ and $\mathbf{p_{i-2}}$. Chef likes these kind of subsets. So he wants to find the size of largest of such subsets.

Input

First line of the input contains one integer **T** - number of tests. **T** tests follows.

First line of each test contains one integer **N** - number of points.

The next **N** lines of each test case contain integer point coordinates $\mathbf{x_i}$, $\mathbf{y_i}$.

Output

For a each test case, output a single integer in separate line corresponding to the largest subset size.

Constraints

- $1 \leq$ **T, N**, sum over all **N** over all all test cases ≤ 2000
- $1 \leq \mathbf{x_i}, \mathbf{y_i} \leq 10^9$

Example

```
Input:
3
3
1 1
4 2
9 3
4
1 1
4 2
5 2
9 3
4
2 3
4 1
1 2
3 2

Output:
3
3
4
```

Explanation

Explanation case 1.

We can get sequence (1, 1) - (4, 2) - (9, 3), point (9, 3) lies below line between point (1, 1) and (4, 2).

Explanation case 2.

We can get (1, 1) -(4, 2) - (9, 3)

Explanation case 3.

We can get (1, 2) -(2, 3) - (3, 2) - (4, 1)

Problem code: KOL16L
Problem name: Optimal BST Revisited

A rooted binary tree is a rooted tree such that each node has atmost 2 children. A Binary Search Tree (BST) is a rooted binary tree, which has values stored in each of its nodes. We are interested only in distinct values. A BST has to satisfy the property that, for every node, its value must be greater than the value of every node on its left sub-tree, if it exists. Also, for every node, its value must be lesser than the value of every node on its right sub-tree, if it exists.

Thunk is bored with the classical problem of finding an optimal BST. He decides to spice it up, and changes the cost function. The problem he is considering is as follows: Given a BST with **N** nodes, called **T**, which store the values **{1,2,..,N}**, and a sequence of Searches **S = (s_1, s_2, ..., s_m)**, he puts his finger on the root of **T**, and then traces the path to the node containing s_1, and then from there, to the node containing s_2, ..., and he finally traces the path from s_{m-1} to s_m. Note that during this process, he might have passed through, or even searched for the same element/node multiple times, if the Search sequence was so given. And he defines the **Cost** to be the total distance (or the number of edges) that he has passed through.

Formally, **Cost(T,S) = Depth(s_1) + Distance(s_1, s_2) + Distance(s_2, s_3) + ... + Distance(s_{m-1}, s_m)**.

Depth(u) is defined to be the number of edges on the unique path from the root of the BST to the node conatining the value **u**. **Distance(u,v)** is defined to be the number of edges on the unique path from the node containing the value **u** to the node containing the value **v**.

Thunk has to hurry to finish his The Art of Competitive Programming series of books, and so doesn't want to waste too much time on this. So, given the sequence of Searches, **S = (s_1, s_2, ..., s_m)**, help him find the minimum **Cost**, over all valid BSTs on **{1,2,..,N}**. That is, find the minimum of **Cost(T,S)**, over all valid **T**s.

Input

The first line contains two integers **N, M**, denoting the number of nodes to be present in the BST, and the length of the search sequence **S**.

The next line contains **M** integers ,separated by single spaces: $s_1, s_2, .., s_M$, denoting search sequence **S**.

Output

Output a single line containing the answer, which is the minimum Cost over all valid BSTs.

Constraints

- $1 \leq N \leq 500$
- $1 \leq M \leq 10^6$
- $1 \leq S_i \leq N$

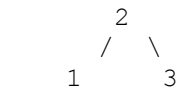
Example

Input:
```
3 5
1 3 2 1 2
```
Output:
```
5
```

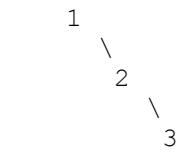
Explanation

For **N** = 3, there are 5 valid BSTs possible:

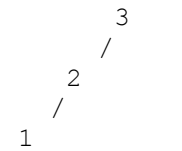
T_1:



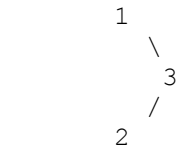
T_2:



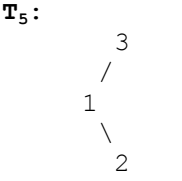
T_3:



T_4:



T_5:



S = (1, 3, 2, 1, 2)

Cost(T_1, S) = Depth(1) + Distance(1, 3) + Distance(3, 2) + Distance(2, 1) + Distance(1, 2)
= 1 + 2 + 1 + 1 + 1 = 6

Cost(T_2, S) = Depth(1) + Distance(1, 3) + Distance(3, 2) + Distance(2, 1) + Distance(1, 2)
= 0 + 2 + 1 + 1 + 1 = 5

Cost(T_3, S) = Depth(1) + Distance(1, 3) + Distance(3, 2) + Distance(2, 1) + Distance(1, 2)
= 2 + 2 + 1 + 1 + 1 = 7

Cost(T_4, S) = Depth(1) + Distance(1, 3) + Distance(3, 2) + Distance(2, 1) + Distance(1, 2)
= 0 + 1 + 1 + 2 + 2 = 6

Cost(T_5, S) = Depth(1) + Distance(1, 3) + Distance(3, 2) + Distance(2, 1) + Distance(1, 2)
= 1 + 1 + 2 + 1 + 1 = 6

We see that the minimum **Cost** is attained by $\mathbf{T_2}$, and it is **5**. Hence the answer is **5**.