Candy

Problem

Sean and Patrick are brothers who just got a nice bag of candy from their parents. Each piece of candy has some positive integer value, and the children want to divide the candy between them. First, Sean will split the candy into two piles, and choose one to give to Patrick. Then Patrick will try to calculate the value of each pile, where the value of a pile is the sum of the values of all pieces of candy in that pile; if he decides the piles don't have equal value, he will start crying.

Unfortunately, Patrick is very young and doesn't know how to add properly. He almost knows how to add numbers in binary; but when he adds two 1s together, he always forgets to carry the remainder to the next bit. For example, if he wants to sum 12 (1100 in binary) and 5 (101 in binary), he will add the two rightmost bits correctly, but in the third bit he will forget to carry the remainder to the next bit:

```
  1100
+ 0101
------
  1001
```

So after adding the last bit without the carry from the third bit, the final result is 9 (1001 in binary). Here are some other examples of Patrick's math skills:

5 + 4 = 1

7 + 9 = 14

50 + 10 = 56

Sean is very good at adding, and he wants to take as much value as he can without causing his little brother to cry. If it's possible, he will split the bag of candy into two non-empty piles such that Patrick thinks that both have the same value. Given the values of all pieces of candy in the bag, we would like to know if this is possible; and, if it's possible, determine the maximum possible value of Sean's pile.

Input

The first line of the input gives the number of test cases, T. T test cases follow. Each test case is described in two lines. The first line contains a single integer N, denoting the number of candies in the bag. The next line contains the N Ci integers separated by single spaces, which denotes the value of each candy in the bag.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1). If it is impossible for Sean to keep Patrick from crying, y should be the word "NO". Otherwise, y should be the value of the pile of candies that Sean will keep.

Limits

$1 \leq T \leq 100$.

$1 \leq Ci \leq 106$.

Small dataset

$2 \leq N \leq 15$.

Large dataset

$2 \leq N \leq 1000$.

Sample

Input

Output

```
2
5
1 2 3 4 5
3
3 5 6
Case #1: NO
Case #2: 11
```

Problem : Robot

Blue and Orange are friendly robots. An evil computer mastermind has locked them up in separate hallways to test them, and then possibly give them cake.

Each hallway contains 100 buttons labeled with the positive integers {1, 2, ..., 100}. Button k is always k meters from the start of the hallway, and the robots both begin at button 1. Over the period of one second, a robot can walk one meter in either direction, or it can press the button at its position. To complete the test, the robots need to push a certain sequence of buttons in a certain order. Both robots know the full sequence in advance. How fast can they complete it?
For example, let's consider the following button sequence:
O 2, B 1, B 2, O 4
Here, O 2 means button 2 in Orange's hallway, B 1 means button 1 in Blue's hallway, and so on. The robots can push this sequence of buttons in 6 seconds using the strategy shown below:

Time	Orange	Blue
1	Move to button 2	Stay at button 1
2	Push button 2	Stay at button 1
3	Move to button 3	Push button 1
4	Move to button 4	Move to button 2
5	Stay at button 4	Push button 2
6	Push button 4	Stay at button 2

Note that Blue has to wait until Orange has completely finished pushing O 2 before it can start pushing B 1.
Input
The first line of the input gives the number of test cases, T. T test cases follow.
Each test case consists of a single line beginning with a positive integer N, representing the number of buttons that need to be pressed. This is followed by N terms of the form "Ri Pi" where Ri is a robot color (always 'O' or 'B'), and Pi is a button position.
Output
For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the minimum number of seconds required for the robots to push the given buttons, in order.
Limits
$1 \leq Pi \leq 100$ for all i.
Small dataset
$1 \leq T \leq 20$.
$1 \leq N \leq 10$.
Large dataset
$1 \leq T \leq 100$.
$1 \leq N \leq 100$.
Sample

Input

Output

```
3
4 O 2 B 1 B 2 O 4
3 O 5 O 8 B 100
2 B 2 B 1
Case #1: 6
Case #2: 100
Case #3: 4
```

Introduction

Magicka™ is an action-adventure game developed by Arrowhead Game Studios. In Magicka you play a wizard, invoking and combining elements to create Magicks. This problem has a similar idea, but it does not assume that you have played Magicka.

Note: "invoke" means "call on." For this problem, it is a technical term and you don't need to know its normal English meaning.

Problem

As a wizard, you can **invoke** eight elements, which are the "base" elements. Each base element is a single character from {Q, W, E, R, A, S, D, F}. When you invoke an element, it gets appended to your **element list**. For example: if you invoke W and then invoke A, (we'll call that "invoking WA" for short) then your element list will be [W, A].

We will specify pairs of base elements that **combine** to form non-base elements (the other 18 capital letters). For example, Q and F might combine to form T. If the two elements from a pair appear at the end of the element list, then both elements of the pair will be immediately removed, and they will be replaced by the element they form. In the example above, if the element list looks like [A, Q, F] or [A, F, Q] at any point, it will become [A, T].

We will specify pairs of base elements that are **opposed** to each other. After you invoke an element, if it isn't immediately combined to form another element, and it is opposed to something in your element list, then your element list will be cleared.

For example, suppose Q and F combine to make T. R and F are opposed to each other. Then invoking the following things (in order, from left to right) will have the following results:

- QF → [T] (Q and F combine to form T)
- QEF → [Q, E, F] (Q and F can't combine because they were never at the end of the element list together)
- RFE → [E] (F and R are opposed, so the list is cleared; then E is invoked)
- REF → [] (F and R are opposed, so the list is cleared)
- RQF → [R, T] (QF combine to make T, so the list is not cleared)
- RFQ → [Q] (F and R are opposed, so the list is cleared)

Given a list of elements to invoke, what will be in the element list when you're done?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of a single line, containing the following space-separated elements in order:

First an integer **C**, followed by **C** strings, each containing three characters: two base elements followed by a non-base element. This indicates that the two base elements combine to form the non-base element. Next will come an integer **D**, followed by **D** strings, each containing two characters: two base elements that are opposed to each other. Finally there will be an integer **N**, followed by a single string containing **N** characters: the series of base elements you are to invoke. You will invoke them in the

order they appear in the string (leftmost character first, and so on).

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is a list in the format "[e_0, e_1, ...]" where e_i is the i^{th} element of the final element list. Please see the sample output for examples.

Limits

$1 \leq \mathbf{T} \leq 100$.
Each pair of base elements may only appear together in one combination, though they may appear in a combination and also be opposed to each other.
No base element may be opposed to itself.
Unlike in the computer game Magicka, there is no limit to the length of the element list.

Small dataset

$0 \leq \mathbf{C} \leq 1$.
$0 \leq \mathbf{D} \leq 1$.
$1 \leq \mathbf{N} \leq 10$.

Large dataset

$0 \leq \mathbf{C} \leq 36$.
$0 \leq \mathbf{D} \leq 28$.
$1 \leq \mathbf{N} \leq 100$.

Sample

Input	Output
5 0 0 2 EA 1 QRI 0 4 RRQR 1 QFT 1 QF 7 FAQFDFQ 1 EEZ 1 QE 7 QEEEERA 0 1 QW 2 QW	Case #1: [E, A] Case #2: [R, I, R] Case #3: [F, D, T] Case #4: [Z, E, R, A] Case #5: []

Problem : goro

Goro has 4 arms. Goro is very strong. You don't mess with Goro. Goro needs to sort an array of **N** different integers. Algorithms are not Goro's strength; strength is Goro's strength. Goro's plan is to use the fingers on two of his hands to hold down several elements of the array and hit the table with his third and fourth fists as hard as possible. This will make the unsecured elements of the array fly up into the air, get shuffled randomly, and fall back down into the empty array locations.

Goro wants to sort the array as quickly as possible. How many hits will it take Goro to sort the given array, on average, if he acts intelligently when choosing which elements of the array to hold down before each hit of the table?

More precisely, before each hit, Goro may choose any subset of the elements of the array to freeze in place. He may choose differently depending on the outcomes of previous hits. Each hit permutes the unfrozen elements uniformly at random. Each permutation is equally likely.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one will consist of two lines. The first line will give the number **N**. The second line will list the **N** elements of the array in their initial order.

Output

For each test case, output one line containing "Case #**x**: **y**", where **x** is the case number (starting from 1) and **y** is the expected number of hit-the-table operations when following the best hold-down strategy. Answers with an absolute or relative error of at most 10^{-6} will be considered correct.

Limits

$1 \leq \mathbf{T} \leq 100$;
The second line of each test case will contain a permutation of the **N** smallest positive integers.
Goro has more than **N** fingers on each hand.

Small dataset

$1 \leq \mathbf{N} \leq 10$;

Large dataset

$1 \leq \mathbf{N} \leq 1000$;

Sample

Input	Output

```
3
2
2 1     Case #1: 2.000000
3       Case #2: 2.000000
1 3 2   Case #3: 4.000000
4
2 1 4 3
```

Explanation

In test case #3, one possible strategy is to hold down the two leftmost elements first. Elements 3 and 4 will be free to move. After a table hit, they will land in the correct order [3, 4] with probability 1/2 and in the wrong order [4, 3] with probability 1/2. Therefore, on average it will take 2 hits to arrange them in the correct order. After that, Goro can hold down elements 3 and 4 and hit the table until 1 and 2 land in the correct order, which will take another 2 hits, on average. The total is then 2 + 2 = 4 hits.