A. Division?

1 second, 256 megabytes

Codeforces separates its users into 4 divisions by their rating:

- For Division 1: $1900 \le \text{rating}$
- For Division 2: $1600 \le \text{rating} \le 1899$
- For Division 3: $1400 \le \text{rating} \le 1599$
- For Division 4: $\text{rating} \le 1399$

Given a rating, print in which division the rating belongs.

Input

The first line of the input contains an integer t ($1 \le t \le 10^4$) — the number of testcases.

The description of each test consists of one line containing one integer rating ($-5000 \le \text{rating} \le 5000$).

Output

For each test case, output a single line containing the correct division in the format "`Division X`", where X is an integer between 1 and 4 representing the division for the corresponding rating.

input

```
7
-789
1299
1300
1399
1400
1679
2300
```

output

```
Division 4
Division 4
Division 4
Division 4
Division 3
Division 2
Division 1
```

For test cases $1-4$, the corresponding ratings are -789, 1299, 1300, 1399, so all of them are in division 4.

For the fifth test case, the corresponding rating is 1400, so it is in division 3.

For the sixth test case, the corresponding rating is 1679, so it is in division 2.

For the seventh test case, the corresponding rating is 2300, so it is in division 1.

B. Triple

1 second, 256 megabytes

Given an array a of n elements, print any value that appears at least three times or print `-1` if there is no such value.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — the elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print any value that appears at least three times or print `-1` if there is no such value.

input

```
7
1
1
3
2 2 2
7
2 2 3 3 4 2 2
8
1 4 3 4 3 2 4 1
9
1 1 1 2 2 2 3 3 3
5
1 5 2 4 3
4
4 4 4 4
```

output

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

In the first test case there is just a single element, so it can't occur at least three times and the answer is `-1`.

In the second test case, all three elements of the array are equal to 2, so 2 occurs three times, and so the answer is 2.

For the third test case, 2 occurs four times, so the answer is 2.

For the fourth test case, 4 occurs three times, so the answer is 4.

For the fifth test case, 1, 2 and 3 all occur at least three times, so they are all valid outputs.

For the sixth test case, all elements are distinct, so none of them occurs at least three times and the answer is `-1`.

C. Number Replacement

2 seconds, 256 megabytes

An integer array $a_1, a_2, \dots, a_n$ is being transformed into an array of lowercase English letters using the following prodecure:

While there is at least one number in the array:

- Choose any number x from the array a, and any letter of the English alphabet y.
- Replace all occurrences of number x with the letter y.

For example, if we initially had an array $a = [2, 3, 2, 4, 1]$, then we could transform it the following way:

- Choose the number 2 and the letter `c`. After that $a = [c, 3, c, 4, 1]$.
- Choose the number 3 and the letter `a`. After that $a = [c, a, c, 4, 1]$.
- Choose the number 4 and the letter `t`. After that $a = [c, a, c, t, 1]$.
- Choose the number 1 and the letter `a`. After that $a = [c, a, c, t, a]$.

After the transformation all letters are united into a string, in our example we get the string "`cacta`".

Having the array a and the string s determine if the string s could be got from the array a after the described transformation?

Input

The first line contains a single integer t ($1 \le t \le 10^3$) — the number of test cases.

Then the description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 50$) — the length of the array a and the string s.

The second line of each test case contains exactly n integers: $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 50$) — the elements of the array a.

The third line of each test case contains a string s of length n, consisting of lowercase English letters.

Output

For each test case, output "YES", if we can get the string s from the array a, and "NO" otherwise. You can output each letter in any case.

input

```
7
5
2 3 2 4 1
cacta
1
50
a
2
11 22
ab
4
1 2 2 1
aaab
5
1 2 3 2 1
aaaaa
6
1 10 2 9 3 8
azzfdb
7
1 2 3 4 1 1 2
abababb
```

output

```
YES
YES
YES
NO
YES
YES
NO
```

The first test case corresponds to the sample described in the statement.

In the second test case we can choose the number 50 and the letter a.

In the third test case we can choose the number 11 and the letter a, after that $a = [a, 22]$. Then we choose the number 22 and the letter b and get $a = [a, b]$.

In the fifth test case we can change all numbers one by one to the letter a.

D. Equal Candies

1 second, 256 megabytes

There are n boxes with different quantities of candies in each of them. The i-th box has a_i candies inside.

You also have n friends that you want to give the candies to, so you decided to give each friend a box of candies. But, you don't want any friends to get upset so you decided to eat some (possibly none) candies from each box so that all boxes have the same quantity of candies in them. Note that you may eat a different number of candies from different boxes and you cannot add candies to any of the boxes.

What's the minimum total number of candies you have to eat to satisfy the requirements?

Input

The first line contains an integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 50$) — the number of boxes you have.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^7$) — the quantity of candies in each box.

Output

For each test case, print a single integer denoting the minimum number of candies you have to eat to satisfy the requirements.

input

```
5
5
1 2 3 4 5
6
1000 1000 5 1000 1000 1000
10
1 2 3 5 1 2 7 9 13 5
3
8 8 8
1
10000000
```

output

```
10
4975
38
0
0
```

For the first test case, you can eat 1 candy from the second box, 2 candies from the third box, 3 candies from the fourth box and 4 candies from the fifth box. Now the boxes have $[1, 1, 1, 1, 1]$ candies in them and you ate $0 + 1 + 2 + 3 + 4 = 10$ candies in total so the answer is 10.

For the second test case, the best answer is obtained by making all boxes contain 5 candies in them, thus eating $995 + 995 + 0 + 995 + 995 + 995 = 4975$ candies in total.

E. Two Groups

1 second, 256 megabytes

You are given an array a consisting of n integers. You want to distribute these n integers into two groups s_1 and s_2 (groups can be empty) so that the following conditions are satisfied:

- For each i ($1 \le i \le n$), a_i goes into exactly one group.
- The value $|sum(s_1)| - |sum(s_2)|$ is the maximum possible among all such ways to distribute the integers.
  Here $sum(s_1)$ denotes the sum of the numbers in the group s_1, and $sum(s_2)$ denotes the sum of the numbers in the group s_2.

Determine the maximum possible value of $|sum(s_1)| - |sum(s_2)|$.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 10^5$) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2 \ldots a_n$ ($-10^9 \le a_i \le 10^9$) — elements of the array a.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer — the maximum possible value of $|sum(s_1)| - |sum(s_2)|$.

input

```
4
2
10 -10
4
-2 -1 11 0
3
2 3 2
5
-9 2 0 0 -4
```

output

```
0
8
7
11
```

In the **first testcase**, we can distribute as $s_1 = \{10\}, s_2 = \{-10\}$. Then the value will be $|10| - |-10| = 0$.

In the **second testcase**, we can distribute as $s_1 = \{0, 11, -1\}, s_2 = \{-2\}$. Then the value will be $|0 + 11 - 1| - |-2| = 10 - 2 = 8$.

In the **third testcase**, we can distribute as $s_1 = \{2, 3, 2\}, s_2 = \{\}$. Then the value will be $|2 + 3 + 2| - |0| = 7$.

In the **fourth testcase**, we can distribute as $s_1 = \{-9, -4, 0\}, s_2 = \{2, 0\}$. Then the value will be $|-9 - 4 + 0| - |2 + 0| = 13 - 2 = 11$.

F. Three Threadlets

2 seconds, 256 megabytes

Once upon a time, bartender Decim found three threadlets and a pair of scissors.

In one operation, Decim chooses any threadlet and cuts it into two threadlets, whose lengths are **positive integers** and their sum is **equal** to the length of the threadlet being cut.

For example, he can cut a threadlet of length 5 into threadlets of lengths 2 and 3, but he cannot cut it into threadlets of lengths 2.5 and 2.5, or lengths 0 and 5, or lengths 3 and 4.

Decim can perform **at most** three operations. He is allowed to cut the threadlets obtained from previous cuts. Will he be able to make all the threadlets of equal length?

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases. Then follows the description of each test case.

In a single line of each test case, there are three integers a, b, c ($1 \le a, b, c \le 10^9$) — the lengths of the threadlets.

Output

For each test case, output "YES" if it is possible to make all the threadlets of equal length by performing at most three operations, otherwise output "NO".

You can output "YES" and "NO" in any case (for example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as a positive answer).

input

```
15
1 3 2
5 5 5
6 36 12
7 8 7
6 3 3
4 4 12
12 6 8
1000000000 1000000000 1000000000
3 7 1
9 9 1
9 3 6
2 8 2
5 3 10
8 4 8
2 8 4
```

output

```
YES
YES
NO
NO
YES
YES
NO
YES
NO
NO
YES
YES
NO
YES
NO
```

Let's consider some testcases of the first test.

In the first testcase, you can apply following operations:

$1, 3, 2 \to 1, 2, 1, 2 \to 1, 1, 1, 1, 2 \to 1, 1, 1, 1, 1, 1$.

In the second testcase, you can do nothing, the threadlets are already of equal length.

In the third testcase, it isn't possible to make threadlets of equal length.

G. Don't Try to Count

2 seconds, 256 megabytes

Given a string x of length n and a string s of length m ($n \cdot m \le 25$), consisting of lowercase Latin letters, you can apply any number of operations to the string x.

In one operation, you append the current value of x to the end of the string x. Note that the value of x will change after this.

For example, if $x =$ "aba", then after applying operations, x will change as follows: "aba" $\to$ "abaaba" $\to$ "abaabaabaaba".

After what **minimum** number of operations s will appear in x as a substring? A substring of a string is defined as a **contiguous** segment of it.

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two numbers n and m ($1 \le n \cdot m \le 25$) — the lengths of strings x and s, respectively.

The second line of each test case contains the string x of length n.

The third line of each test case contains the string s of length m.

Output

For each test case, output a single number — the **minimum** number of operations after which s will appear in x as a substring. If this is not possible, output -1.

input

```
12
1 5
a
aaaaa
5 5
eforc
force
2 5
ab
ababa
3 5
aba
ababa
4 3
babb
bbb
5 1
aaaaa
a
4 2
aabb
ba
2 8
bk
kbkbkbkb
12 2
fjdgmujlcont
tf
2 2
aa
aa
3 5
abb
babba
1 19
m
mmmmmmmmmmmmmmmmmmm
```

output

```
3
1
2
-1
1
0
1
3
1
0
2
5
```

In the first test case of the example, after 2 operations, the string will become "aaaa", and after 3 operations, it will become "aaaaaaaa", so the answer is 3.

In the second test case of the example, after applying 1 operation, the string will become "eforceforc", where the substring is highlighted in red.

In the fourth test case of the example, it can be shown that it is impossible to obtain the desired string as a substring.

H. Goals of Victory

1 second, 256 megabytes

There are n teams in a football tournament. Each pair of teams match up once. After every match, Pak Chanek receives two integers as the result of the match, the number of goals the two teams score during the match. The efficiency of a team is equal to the total number of goals the team scores in each of its matches minus the total number of goals scored by the opponent in each of its matches.

After the tournament ends, Pak Dengklek counts the efficiency of every team. Turns out that he forgot about the efficiency of one of the teams. Given the efficiency of $n-1$ teams $a_1, a_2, a_3, \ldots, a_{n-1}$. What is the efficiency of the missing team? It can be shown that the efficiency of the missing team can be uniquely determined.

Input

Each test contains multiple test cases. The first line contains an integer t ($1 \le t \le 500$) — the number of test cases. The following lines contain the description of each test case.

The first line contains a single integer n ($2 \le n \le 100$) — the number of teams.

The second line contains $n-1$ integers $a_1, a_2, a_3, \ldots, a_{n-1}$ ($-100 \le a_i \le 100$) — the efficiency of $n-1$ teams.

Output

For each test case, output a line containing an integer representing the efficiency of the missing team.

input

```
2
4
3 -4 5
11
-30 12 -57 7 0 -81 -68 41 -89 0
```

output

```
-4
265
```

In the first test case, below is a possible tournament result:

- Team 1 vs. Team 2: $1-2$
- Team 1 vs. Team 3: $3-0$
- Team 1 vs. Team 4: $3-2$
- Team 2 vs. Team 3: $1-4$
- Team 2 vs. Team 4: $1-3$
- Team 3 vs. Team 4: $5-0$

The efficiency of each team is:

1. Team 1: $(1+3+3)-(2+0+2)=7-4=3$
2. Team 2: $(2+1+1)-(1+4+3)=4-8=-4$
3. Team 3: $(0+4+5)-(3+1+0)=9-4=5$
4. Team 4: $(2+3+0)-(3+1+5)=5-9=-4$

Therefore, the efficiency of the missing team (team 4) is -4.

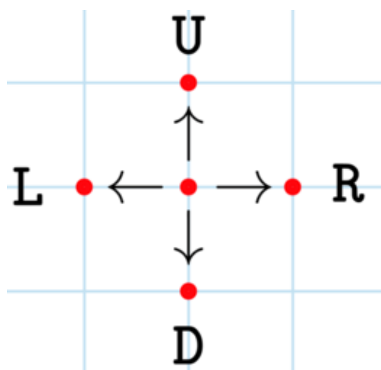
It can be shown that any possible tournament of 4 teams that has the efficiency of 3 teams be 3, -4, and 5 will always have the efficiency of the 4-th team be -4.

I. Following Directions

1 second, 256 megabytes

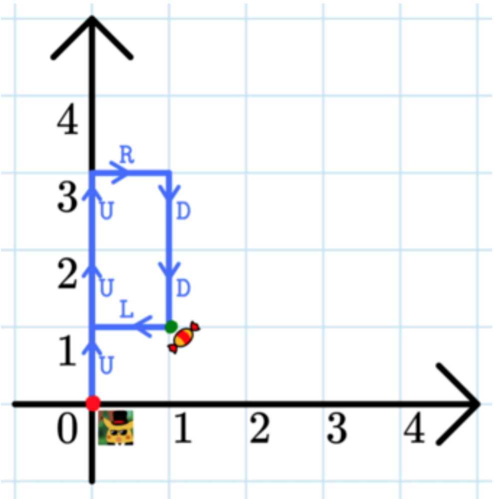
Alperen is standing at the point $(0, 0)$. He is given a string s of length n and performs n moves. The i-th move is as follows:

- if $s_i = \texttt{L}$, then move one unit left;
- if $s_i = \texttt{R}$, then move one unit right;
- if $s_i = \texttt{U}$, then move one unit up;
- if $s_i = \texttt{D}$, then move one unit down.



If Alperen starts at the center point, he can make the four moves shown.

There is a candy at $(1, 1)$ (that is, one unit above and one unit to the right of Alperen's starting point). You need to determine if Alperen ever passes the candy.



Alperen's path in the first test case.

Input

The first line of the input contains an integer t ($1 \le t \le 1000$) — the number of testcases.

The first line of each test case contains an integer n ($1 \le n \le 50$) — the length of the string.

The second line of each test case contains a string s of length n consisting of characters `L`, `R`, `D`, and `U`, denoting the moves Alperen makes.

Output

For each test case, output "YES" (without quotes) if Alperen passes the candy, and "NO" (without quotes) otherwise.

You can output the answer in any case (for example, the strings "yEs", "yes", "Yes" and "YES" will be recognized as a positive answer).

input

```
7
7
UUURDDL
2
UR
8
RRRUUDDD
3
LLL
4
DUUR
5
RUDLL
11
LLLLDDRUDRD
```

output

```
YES
YES
NO
NO
YES
YES
NO
```

In the first test case, Alperen follows the path

$$(0,0) \xrightarrow{\texttt{U}} (0,1) \xrightarrow{\texttt{U}} (0,2) \xrightarrow{\texttt{U}} (0,3) \xrightarrow{\texttt{R}} (1,3) \xrightarrow{\texttt{D}} (1,2) \xrightarrow{\texttt{D}} \mathbf{(1,1)} \xrightarrow{\texttt{L}} (0,1).$$

Note that Alperen doesn't need to end at the candy's location of $(1,1)$, he just needs to pass it at some point.

In the second test case, Alperen follows the path

$$(0,0) \xrightarrow{\texttt{U}} (0,1) \xrightarrow{\texttt{R}} \mathbf{(1,1)}.$$

In the third test case, Alperen follows the path

$$(0,0) \xrightarrow{\texttt{R}} (1,0) \xrightarrow{\texttt{R}} (2,0) \xrightarrow{\texttt{R}} (3,0) \xrightarrow{\texttt{U}} (3,1) \xrightarrow{\texttt{U}} (3,2) \xrightarrow{\texttt{D}} (3,1) \xrightarrow{\texttt{D}} (3,0) \xrightarrow{\texttt{D}} (3,-1).$$

In the fourth test case, Alperen follows the path

$$(0,0) \xrightarrow{\texttt{L}} (-1,0) \xrightarrow{\texttt{L}} (-2,0) \xrightarrow{\texttt{L}} (-3,0).$$

J. FashionabLee

2 seconds, 256 megabytes

Lee is going to fashionably decorate his house for a party, using some regular convex polygons...

Lee thinks a regular n-sided (convex) polygon is *beautiful* if and only if he can rotate it in such a way that at least one of its edges is parallel to the OX-axis and at least one of its edges is parallel to the OY-axis at the same time.

Recall that a regular n-sided polygon is a convex polygon with n vertices such that all the edges and angles are equal.

Now he is shopping: the market has t regular polygons. For each of them print YES if it is beautiful and NO otherwise.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of polygons in the market.

Each of the next t lines contains a single integer n_i ($3 \le n_i \le 10^9$): it means that the i-th polygon is a regular n_i-sided polygon.

Output

For each polygon, print YES if it's beautiful or NO otherwise (case insensitive).

input

```
4
3
4
12
1000000000
```

output

```
NO
YES
YES
YES
```

In the example, there are 4 polygons in the market. It's easy to see that an equilateral triangle (a regular 3-sided polygon) is not beautiful, a square (a regular 4-sided polygon) is beautiful and a regular 12-sided polygon (is shown below) is beautiful as well.