

## Precise Average 2 (avg2)

Your friend John still works at the same shop and it still sells  $N$  products, numbered from 0 to  $N - 1$ . Product  $i$  has a price of  $P_i$  bytedollars, where  $P_i$  is a positive integer.

A recent Byteland law stated that the average of those prices must be exactly  $K$ , a positive integer. John has already tried to change the prices to comply with the law, but the boss was not satisfied with the result, since some prices changed too much. So he asked John for another way to change the prices!




Figure 1: John's angry boss.

John is still busy, so he asked for your help: what is the minimum value of  $C$  for which it is possible to change each price  $P_i$  by *at most*  $C$  such that the average of the new prices is  $K$ ?

More formally, find the minimum value of  $C$  such that for the array of positive integers  $P = [P_0, P_1, \dots, P_{N-1}]$  there exists an array of positive integers  $P' = [P'_0, P'_1, \dots, P'_{N-1}]$  such that:

- the average of the elements of  $P'$  is  $K$ , and
- $|P'_i - P_i| \leq C$  for each  $0 \leq i < N$ .

It can be proven that such a  $C$  always exists.

 Among the attachments of this task you may find a template file `avg2.*` with a sample incomplete implementation.

### Input

The first line contains the integers  $N$  and  $K$ . The second line contains  $N$  integers  $P_i$ .

### Output

You need to write a single line with the integer  $C$ .

## Constraints

- $1 \leq N \leq 200\,000$ .
- $1 \leq K \leq 1\,000\,000\,000$ .
- $1 \leq P_i \leq 1\,000\,000\,000$  for each  $i = 0 \dots N - 1$ .

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.
- **Subtask 2** (10 points)       $N \leq 2$ .
- **Subtask 3** (25 points)       $K \leq 10$  and  $P_i \leq 10$  for each  $i = 0 \dots N - 1$ .
- **Subtask 4** (24 points)       $N \leq 1000$ .
- **Subtask 5** (41 points)      No additional limitations.

## Examples

input	output
2 5 13 3	4
7 4 3 6 5 6 6 3 9	2

## Explanation

In the **first sample case** the answer is 4. One possible array  $P'$  is  $[9, 1]$ . Note that its average is  $\frac{9+1}{2} = 5$ ,  $|P'_0 - P_0| = |9 - 13| = 4 \leq 4$  and  $|P'_1 - P_1| = |1 - 3| = 2 \leq 4$ . It can be proven that for each  $C < 4$  no suitable  $P'$  exists.

In the **second sample case**, the answer is 2. One possible array  $P'$  is  $[3, 4, 4, 4, 4, 2, 7]$ .

## Farmula 1 (formula1)

*It's lights out and away we go - David Croft*

You became a fan of Formula 1 this year and you decided to invent your own championship where the fastest vehicles can get glory, trophies and wins – except the only vehicles eligible to compete are tractors!

Since you want to make this event as exciting as possible, you decided to name your championship Farmula 1, to give watchers a better idea of what to expect - a combination of circuits and tractors. Now, you seek to follow your driver friend Daniel's results throughout a season, without looking at the results of other drivers. You want to find out whether Daniel did well enough to win the Farmula 1 championship, however the other drivers performed.

The season has  $N$  races and there are 20 tractors registered for the championship. You also know Daniel's placement in each of the races. Specifically, for each race we codify the result as the final position of the driver at the end of the race, an integer between 1 and 20 (for simplicity, we assume that everyone finishes every race).



Figure 1: A very fast tractor.

Given  $N$  numbers encoding Daniel's results, your task is to decide whether he won the championship, however the other drivers performed!

### Scoring system of Farmula 1

For each race, the top ten drivers get points as follows:

- the first place gets 25 points,
- the second place gets 18 points,
- the third place gets 15 points,
- the fourth place gets 12 points,
- the fifth place gets 10 points,
- the sixth place gets 8 points,
- the seventh place gets 6 points,
- the eighth place gets 4 points,
- the ninth place gets 2 points, and
- the tenth place gets 1 point.

If there is a tie for the first place at the end of the season, it is resolved as a championship win for all drivers in question.

📎 Among the attachments of this task you may find a template file `formula1.*` with a sample incomplete implementation.

## Input

On the first line you are given  $T$ , the number of test cases.

Each test case consists of two lines. On the first line you are given  $N$ , the number of races. On the next line you are given  $N$  integers  $P_i$ , the  $i$ -th of them being the position Daniel got in the  $i$ -th race.

## Output

For each testcase, print a single line containing the message **Champion**, if the driver did well enough to be a champion, or **Practice harder** otherwise.

## Constraints

- $1 \leq T \leq 1000$ .
- $1 \leq N \leq 100$ .
- $1 \leq P_i \leq 20$  for each  $i = 0 \dots N - 1$ .

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.  
🏆🏆🏆🏆🏆
- **Subtask 2** (30 points)       $N \leq 5$ , that is, there are at most 5 races in a season.  
🏆🏆🏆🏆🏆
- **Subtask 3** (30 points)      Daniel finished in first or second place in all races.  
🏆🏆🏆🏆🏆
- **Subtask 4** (40 points)      No additional limitations.  
🏆🏆🏆🏆🏆

Examples

input	output
4	Champion
	Practice harder
5	Champion
1 2 2 1 1	Practice harder
5	
8 1 1 2 3	
4	
2 2 1 1	
9	
5 11 3 1 1 4 6 2 1	

Explanation

In the **first sample case**, the driver won the championship irrespective of the results of other drivers: they earned  $25 + 18 + 18 + 25 + 25 = 111$  points in total. It can be proven that no other driver can achieve a better score.

In the **second sample case** Daniel has a score of

$$4 + 25 + 25 + 18 + 15 = 87$$

it is possible for another driver to win the championship, for example, by placing first in the fourth race and second in every other race. This would grant them a score of  $18 + 18 + 18 + 25 + 18 = 97$ .

In the **third sample case**, the driver earned  $18 + 18 + 25 + 25 = 86$  points. It can be proven that the maximum possible total score that a different driver may earn over the season is 86, resulting in a tie. According to the rules, the championship title would awarded to both drivers in this scenario.

## The Jeweller's Game (gemgame)

John is playing the new hit mobile game: “The Jeweller’s Game”.

In this game, there is an  $N \times N$  board full of different kinds of gems. Let’s denote by  $(r, c)$  the cell located at the  $r$ -th row and  $c$ -th column of the board. Each cell of the board contains a gem. The type of the gem in cell  $(r, c)$  is represented by a positive integer  $G_{r,c}$ .

We group the cells according to the following rule. Cells  $a$  and  $b$  are in the same group if and only if there exists a sequence of cells  $p_0, \dots, p_k$  such that:

- $p_0 = a$  and  $p_k = b$ , and
- cells  $p_{i-1}$  and  $p_i$  are edge-adjacent and are of the same type for each  $i = 1, \dots, k$ .


Note that every cell belongs to exactly one group.

The player can improve their score by swapping edge-adjacent cells of the board. Depending on whether the two swapped cells are in the same row or in the same column, we call a swap either horizontal or vertical, respectively. If the two swapped gems are of the same type, then the score of the swap is 0. Otherwise, consider the board **after** performing the swap: the score is the product of the *values* of the two swapped cells. The *value* of a cell is the number of cells in its group (including itself).



Figure 1: There were many similar games in the past.

Find the score of all horizontal and vertical swaps on the board!

 Among the attachments of this task you may find a template file `gemgame.*` with a sample incomplete implementation.

### Input

The input file consists of:

- a line containing integer  $N$ .
- $N$  lines, the  $j$ -th of which consisting of the  $N$  integers  $G_{j,1}, \dots, G_{j,N}$ .

## Output

The first  $N$  lines of the output should contain  $N - 1$  integers each. On the  $i$ -th line the  $j$ -th number ( $1 \leq i \leq N, 1 \leq j < N$ ) should be the score of the swap of cells  $(i, j)$  and  $(i, j + 1)$ .





The next  $N - 1$  lines of the output should contain  $N$  integers each. The  $j$ -th number on  $i$ -th line ( $1 \leq i < N, 1 \leq j \leq N$ ) should be the score of the swap of cells  $(i, j)$  and  $(i + 1, j)$ .

## Constraints

- $2 \leq N \leq 1000$ .
- $1 \leq G_{r,c} \leq 1\,000\,000$  for each  $r = 1 \dots N$  and  $c = 1 \dots N$ .

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.  

- **Subtask 2** (15 points)       $N = 2$ .  

- **Subtask 3** (45 points)       $N \leq 75$ .  

- **Subtask 4** (40 points)      No additional limitations.  


## Examples

input	output
3 1 2 1 1 3 2 2 2 2	2 15 1 5 0 0 0 5 2 1 4 0
4 2 1 9 1 1 2 1 1 2 1 2 7 2 9 2 1	4 4 4 24 12 0 8 16 1 3 3 1 8 12 4 0 6 30 4 2 0 1 0 4

## Explanation


In the **first sample case**, consider the result of the (horizontal) swap of cells  $(1, 2)$  and  $(1, 3)$ :

1	1	2
1	3	2
2	2	2

The groups of the two swapped cells are marked with red and blue. So the score of the swap is  $3 \cdot 5 = 15$ .

## Increasing XOR (increasingxor)

An array  $B$  consisting of  $k$  positive integers is *beautiful* if and only if there exists an array  $C = [C_0, C_1, \dots, C_{k-1}]$  such that  $C$  is a permutation of  $B$  and the sequence of its prefix-XORs is strictly increasing. That is,  $P_0 < P_1 < \dots < P_{k-1}$  where  $P_i = C_0 \oplus C_1 \oplus \dots \oplus C_i$  for each  $i = 0, \dots, k-1$ .

 The bitwise XOR  $a \oplus b$  of two integers  $a$  and  $b$  is defined in the following way: the  $i$ -th bit of  $a \oplus b$  is 1 if and only if exactly one among  $a$  and  $b$  has 1 in the  $i$ -th bit.

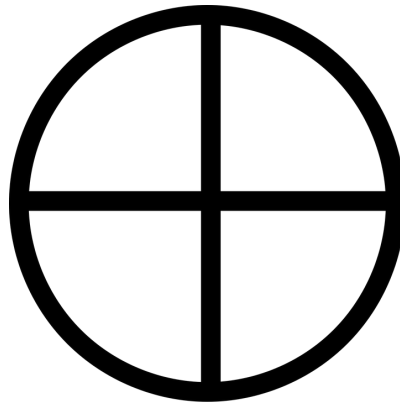



Figure 1: The XOR symbol.

Given an array  $A$  consisting of  $N$  positive integers, you have to determine for each non-empty prefix of  $A$  whether it is *beautiful*.

 Among the attachments of this task you may find a template file `increasingxor.*` with a sample incomplete implementation.

### Input

The first line contains the only integer  $N$ . The second line contains  $N$  integers,  $A_0, A_1, \dots, A_{N-1}$ .

### Output

You should print  $N$  lines. In the  $i$ th line, you must write **YES** if the  $i$ th prefix is *beautiful* and **NO** otherwise.






### Constraints

- $1 \leq N \leq 200\,000$ .
- $1 \leq A_i < 2^{30}$  for each  $i = 0 \dots N-1$ .



# Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- Subtask 1 (0 points)      Examples.  

- Subtask 2 (10 points)       $N \leq 10$ .  

- Subtask 3 (11 points)       $A_i \leq 7$  for each  $i = 0 \dots N - 1$ .  

- Subtask 4 (35 points)       $N \leq 500$ .  

- Subtask 5 (44 points)      No additional limitations.  


# Examples

input	output
5 3 1 4 1 5	YES YES YES YES NO
5 3 3 5 8 19	YES NO NO NO YES

# Explanation

In the **first sample case**:

1. The 1st prefix is *beautiful* because a sequence consisting of a single element is, by definition, strictly increasing.
2. For the 2nd prefix, the permutation 1, 3 is suitable as the sequence  $1, 2 = 1 \oplus 3$  is strictly increasing.
3. For the 3rd prefix, the permutation 1, 3, 4 is suitable as the sequence  $1, 2 = 1 \oplus 3, 6 = 1 \oplus 3 \oplus 4$  is strictly increasing.
4. For the 4th prefix, the permutation 1, 3, 4, 1 is suitable as the sequence 1, 2, 6, 7 is strictly increasing.
5. It can be checked that the 5th prefix is not *beautiful*.

## Good Intervals (intervals)

For some  $k \geq 1$ , we say that the sequence of integers  $v = [v_1, v_2, \dots, v_k]$  is **good** if  $v_i$  is divisible by  $i$  for every  $i$  from 1 to  $k$ .

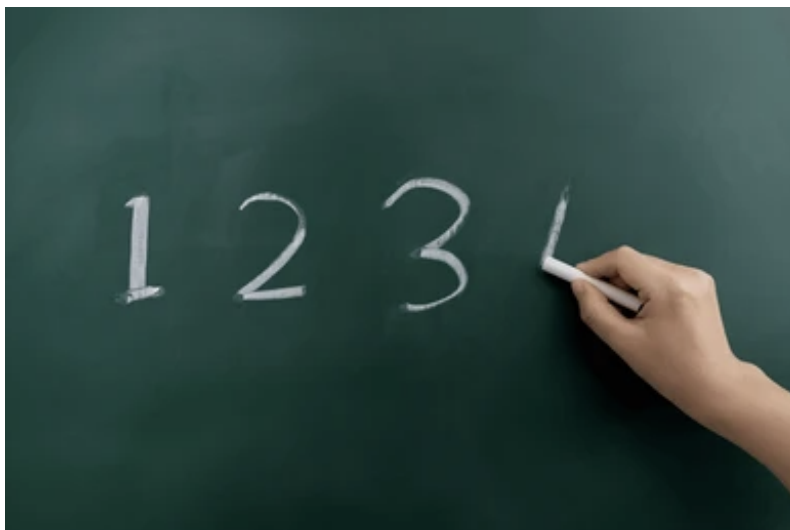


Figure 1: Writing good sequences is a fashionable pastime!

You are given an integer sequence  $A = [A_1, A_2, \dots, A_N]$  of length  $N$  and  $Q$  queries of the form  $l_i, r_i$ , which represent the range of values  $[A_{l_i}, A_{l_i+1}, \dots, A_{r_i}]$  from the sequence. For each query compute the number of subintervals of the given range which are **good** sequences (each subinterval is considered as an independent sequence, indexed from 1).

📎 Among the attachments of this task you may find a template file `intervals.*` with a sample incomplete implementation.

### Input

The first line of the input file contains a single integer  $T$ , the number of test cases.  $T$  test cases follow, each preceded by an empty line.

The first line of each test case contains a single integer  $N$ , the length of the sequence.

The second line of each test case contains  $N$  space-separated integers  $A_i$ , the elements of the sequence.

The third line of each test case contains a single integer  $Q$ , the number of queries.

Each of the following  $Q$  lines contain two integers  $l_i, r_i$ , the query intervals.

### Output






For each test case, output all query results, each on a separate line.

## Constraints

- $1 \leq T \leq 10$ .
- $1 \leq N, Q \leq 100\,000$ .
- $1 \leq A_i \leq 10^{18}$ .
- $1 \leq l_i \leq r_i \leq N$ .
- The sum of  $N$  and  $Q$  over all test cases does not exceed 100 000.

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.  

- **Subtask 2** (10 points)       $N, Q \leq 200$ .  

- **Subtask 3** (25 points)       $N \leq 2000$ .  

- **Subtask 4** (30 points)       $A_i \leq 10^6$  for every  $i = 1 \dots N$ .  

- **Subtask 5** (35 points)      No additional limitations.  


## Examples

input	output
1  5 6 24 6 8 10 3 1 3 2 4 1 5	6 5 12

## Explanation

In the **sample case**, consider the the second query corresponding to the range of values  $[24, 6, 8]$ .

- An example of a good subinterval is  $[A_2, A_3] = [24, 6]$ , because 24 is divisible by 1 and 6 is divisible by 2.
- An example of a subinterval which is not good is  $[A_2, A_3, A_4] = [24, 6, 8]$ , as 8 is not divisible by 3.

The 5 good subintervals for this query are the sequences  $[24]$ ,  $[24, 6]$ ,  $[6]$ ,  $[6, 8]$  and  $[8]$ .

## Pac-Man (pacman)

Alessandro is coding his new 3D version of Pac-Man. The game board is described by a three-dimensional grid, some cells are blocked and the others are free. Pac-Man and the ghosts can move to any free cell sharing a face with the current cell.

Alessandro instructed ghosts to move in a very simple way. If a ghost wants to move from cell  $A$  to cell  $B$ , it will repeat the following procedure until it reaches its destination or fails:

- If the ghost can decrease the distance along the  $x$  axis (i.e.  $|A_x - B_x|$ ) it will do so by moving one cell along the  $x$  axis;
- otherwise if it can decrease the distance along the  $y$  axis (i.e.  $|A_y - B_y|$ ) it will do so by moving one cell along the  $y$  axis;
- otherwise if it can decrease the distance along the  $z$  axis (i.e.  $|A_z - B_z|$ ) it will do so by moving one cell along the  $z$  axis;
- otherwise it fails.

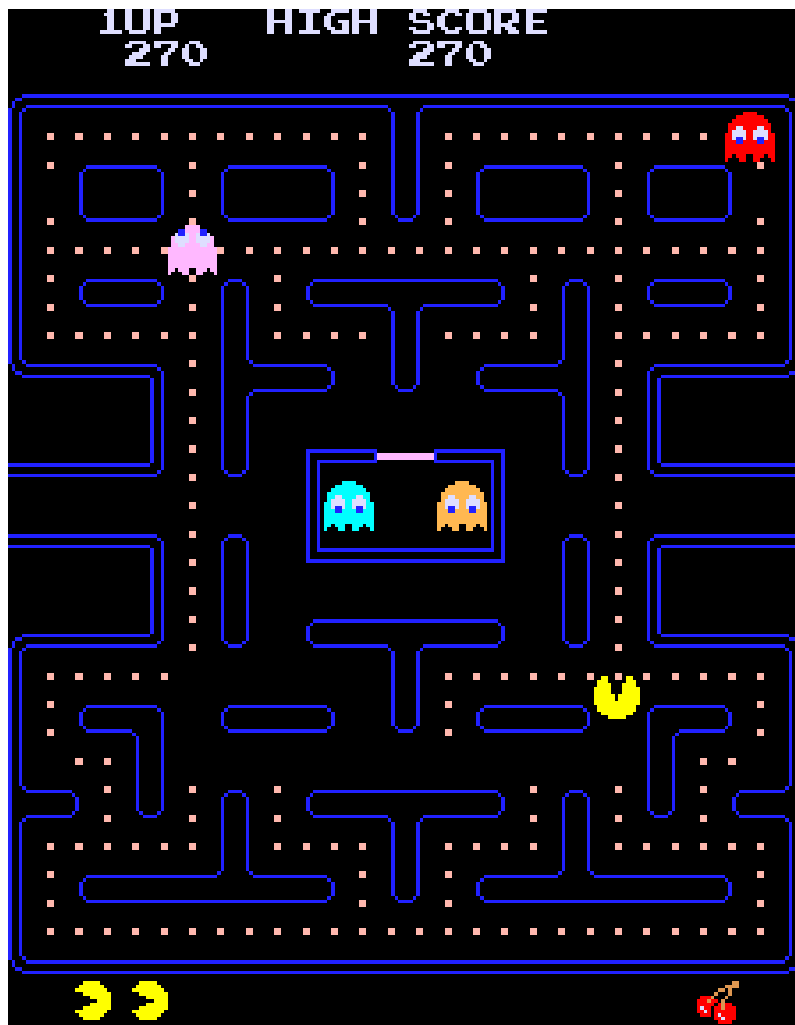


Figure 1: Alessandro playing a classic game of Pac-Man.

Alessandro has three arrays  $X_i$ ,  $Y_i$  and  $Z_i$  indexed from 0 to  $N - 1$  that describe the coordinates of the free cells. He is wondering whether the strategy above is smart enough to control the ghosts.

Help him by writing a program that determines whether for every pair of cells  $A$  and  $B$  a ghost will succeed in reaching cell  $B$  starting from cell  $A$ !

📎 Among the attachments of this task you may find a template file `pacman.*` with a sample incomplete implementation.

## Input

The input file consists of:

- a line containing integer  $N$ .
- a line containing the  $N$  integers  $X_0, \dots, X_{N-1}$ .
- a line containing the  $N$  integers  $Y_0, \dots, Y_{N-1}$ .
- a line containing the  $N$  integers  $Z_0, \dots, Z_{N-1}$ .

## Output







The output file must contain a single line consisting of **YES** if ghosts will always succeed in reaching their destinations or **NO** otherwise.

## Constraints

- $1 \leq N \leq 100\,000$ .
- $0 \leq X_i, Y_i, Z_i < 100\,000$  for each  $i = 0 \dots N - 1$ .

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- |   |   |
|---|---|
| – Subtask 1 (0 points)  | Examples.                                 |
|  |   |
| – Subtask 2 (18 points)   | $N \leq 100$ and $X_i, Y_i, Z_i < 100$ .  |
|  |   |
| – Subtask 3 (19 points)   | $N \leq 7500$ and $X_i, Y_i, Z_i < 100$ . |
|  |   |
| – Subtask 4 (24 points)   | $N \leq 1000$ and $Z_i = 0$ .             |
|  |   |
| – Subtask 5 (22 points)   | $X_i, Y_i, Z_i < 100$ .                   |
|  |   |
| – Subtask 6 (17 points)   | No additional limitations.                |
|  |   |

## Examples

input	output
4 0 0 1 1 0 1 1 2 0 0 0 0	YES
8 0 1 2 2 2 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1	NO
5 0 0 1 1 2 0 1 1 0 2 0 0 0 0 2	NO

## Explanation

In the **first sample case**, ghosts can always reach their destination. For example, a ghost can move from cell  $A = (0, 0, 0)$  to cell  $B = (1, 2, 0)$  following path  $(0, 0, 0) \rightarrow (0, 1, 0) \rightarrow (1, 1, 0) \rightarrow (1, 2, 0)$ .

In the **second sample case**, ghost can't move from cell  $A = (1, 0, 0)$  to cell  $B = (1, 1, 1)$ .

## String Imbalance (stringimbalance)

Bogdan is giving Carlo orthography lessons! The course will be held in  $Q$  days, during which Carlo will have to copy the string written on the blackboard as an exercise. The string is initially empty, and each day  $i$  ( $0 \leq i < Q$ ) will be extended by Bodgan with  $F_i$  copies of the character  $C_i$  that is the lesson's topic. More formally, the exercise string  $S_i$  will be as follows:

- $S_0$  consists of  $F_0$  characters equal to  $C_0$ ;
- $S_i$  is the concatenation of  $S_{i-1}$  and  $F_i$  characters equal to  $C_i$ .

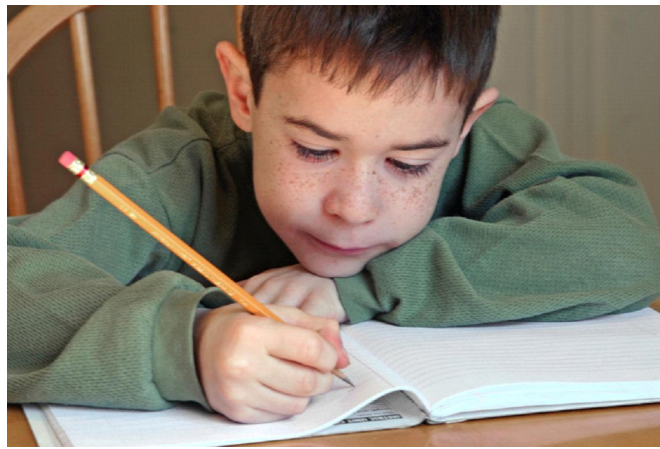



Figure 1: Carlo practising.

Carlo is very prone to mistakes and on day  $i$  he will get up to  $K_i$  letters wrong. Thus, the string he will write on that day will differ from  $S_i$  in at most  $K_i$  positions. Bogdan would like to know the minimum possible *imbalance* of the strings Carlo will write. We define the *imbalance* of a string of length  $N$  as the number of pair of indices  $(i, j)$  such that  $0 \leq i < j < N$  and  $S_i \neq S_j$ .

 Among the attachments of this task you may find a template file `stringimbalance.*` with a sample incomplete implementation.

### Input

The first line of the input file contains a single integer  $T$ , the number of test cases.  $T$  test cases follow, each preceded by an empty line. Each test case consists of:

- a line containing integer  $Q$ .
- $Q$  lines, the  $i$ -th of which consisting of integer  $F_i$ ; character  $C_i$  and integer  $K_i$ .

### Output






For each query from each testcase, output a single line containing one integer denoting the answer for that query.

## Constraints

- $1 \leq T \leq 10$ .
- $1 \leq Q \leq 200\,000$ .
- $1 \leq F_i \leq 5000$ .
- $1 \leq K_i \leq 10^9$ .
- $C_i$  is a lowercase or uppercase letter of the English alphabet.
- The sum of  $Q$  over all test cases does not exceed 200 000.

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.  

- **Subtask 2** (20 points)       $Q \leq 200$  and  $F_i = 1, K_i \leq 10$  for every  $0 \leq i < Q$ .  

- **Subtask 3** (24 points)       $C_i = \text{'a'}$  or  $C_i = \text{'b'}$  for every  $0 \leq i < Q$ .  

- **Subtask 4** (26 points)       $C_i$  is a lowercase letter for every  $0 \leq i < Q$ .  

- **Subtask 5** (30 points)      No additional limitations.  


## Examples

input	output
2	0
	0
3	10
2 a 0	0
3 b 3	4
2 c 2	
2	
2 A 10	
3 a 1	

## Explanation

In the **first testcase of the sample case**:

- $S_0 = \text{"aa"}$ . Its *imbalance* is 0 and it is possible for Carlo to copy it correctly.
- $S_1 = \text{"aabbbb"}$ . It is possible for Carlo to write  $\text{"aaaaa"}$ , which has an imbalance of 0.
- $S_2 = \text{"aabbbscc"}$ . Its imbalance is 16. It is possible for Carlo to write  $\text{"aabbsbbb"}$ , which has an imbalance of 10. He cannot write a string with a lower *imbalance* with up to 2 mistakes.



## Swapping Digits (swappingdigits)

You are given a positive integer  $N$ . In one move you are allowed to swap any two digits of  $N$ .

For example, if  $N$  is 1234, then after swapping the first and the third digit  $N$  becomes 3214, and if  $N$  is 20005, after swapping the first and the fourth digit  $N$  becomes 00025 = 25. Note that you can not get 20005 from 25 (i.e., you should not assume the existence of leading zeroes).

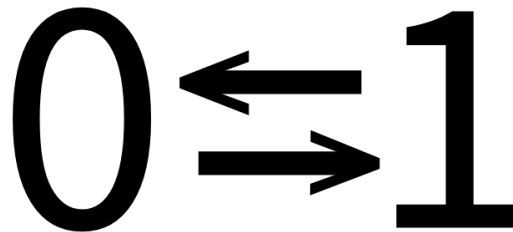



Figure 1: Swapping digits.

What is the minimum number of moves to make  $N$  divisible by 25? If it is not possible to make  $N$  divisible by 25 using the above operation some (possibly zero) number of times, then the answer is  $-1$ .

 Among the attachments of this task you may find a template file `swappingdigits.*` with a sample incomplete implementation.

### Input

The first line of input contains an integer  $T$  denoting the number of testcases.

Each of the next  $T$  lines contains a single integer  $N$ .

### Output

You should print  $T$  lines, the  $i$ -th of which containing a single integer, the answer to the  $i$ -th testcase.

### Constraints

- $1 \leq T \leq 100\,000$ .
- $1 \leq N < 10^{100\,000}$ .
- The total number of digits of  $N$  over all testcases will not exceed 1 000 000.

# Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.
- **Subtask 2** (21 points)       $N \leq 100\,000$ . It is always possible to make  $N$  divisible by 25.
- **Subtask 3** (23 points)       $N \leq 10^9$ .
- **Subtask 4** (41 points)       $N < 10^{1000}$  and the total number of digits of  $N$  over all testcases will not exceed 10 000.
- **Subtask 5** (15 points)      No additional constraints.

# Examples

input	output
2	1
1010	0
25	

# Explanation

In the **first testcase**, you can swap the second and the third digits to make  $N = 1100$  which is divisible by 25. It is impossible to make  $N$  divisible by 25 using fewer moves.

In the **second testcase**,  $N$  is already divisible by 25, so no moves are needed.

## Tulip Bouquets (tulips)

Anna really likes tulips. She has  $N$  tulips in her garden, numbered from  $0$  to  $N - 1$ . The *beautiness* of tulip  $i$  is  $T_i$ . She wants to create  $K$  (non-empty) bouquets from the tulips. To do that, she starts to walk from the first tulip towards the last. At each flower, she can either


- insert it into the current bouquet, or
- finish the current bouquet and start a new one. The current tulip is added to the new bouquet.

Note that, after finishing a bouquet she won't be able to insert more tulips into it!



Figure 1: Tulips are indeed beautiful.

The *beautiness* of a bouquet is the minimum of the *beautinesses* of the tulips in it. She wants to maximize the sum of the *beautinesses* of the  $K$  bouquets by partitioning the tulips optimally. Your task is to calculate this maximum value!

 Among the attachments of this task you may find a template file `tulips.*` with a sample incomplete implementation.

### Input

The input file consists of:

- a line containing integers  $N$ ,  $K$ .
- a line containing the  $N$  integers  $T_0, \dots, T_{N-1}$ .

### Output






The output file must contain a single line with an integer  $M$ , the maximum total *beautiness* of the bouquets.

## Constraints

- $1 \leq K \leq N \leq 100\,000$ .
- $1 \leq N \cdot K \leq 50\,000\,000$ .
- $0 \leq T_i \leq 1\,000\,000\,000$  for each  $i = 0 \dots N - 1$ .

## Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points)      Examples.  

- **Subtask 2** (12 points)       $K \leq 2$ .  

- **Subtask 3** (25 points)       $N \leq 16$ .  

- **Subtask 4** (25 points)       $N \leq 500$ .  

- **Subtask 5** (38 points)      No additional limitations.  


## Examples

input	output
5 2 3 4 1 5 2	4
6 4 4 2 6 1 3 5	14

## Explanation

In the **first sample case**  $3\,4\,|\,1\,5\,2$  is an optimal way to distribute the flowers into 2 bouquets. The total *beautiness* is  $3 + 1 = 4$ .

In the **second sample case**  $4\,2\,|\,6\,|\,1\,3\,|\,5$  is an optimal way to distribute the flowers into 4 bouquets. The total *beautiness* is  $2 + 6 + 1 + 5 = 14$ .