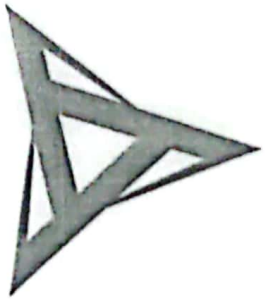
Welcome to PUCon'24

"First, solve the problem. Then, write the code."

– John Johnson



PUCon' 24

Final Round

Programming Competition

FCIT

University of the Punjab

Problem A

Bazinga!

As part of your Mobile Development semester project, you've developed a new game where players are presented with two words. If these words can be transformed into one another, players must press the button labeled "Bazinga"; otherwise, they press "Pass". The first person to correctly press either of these buttons earns points. The rules for "Bazinga" are as follows:

1. **Operation 1**: Swap any two existing characters
   For example, abcde -> aecdb
2. **Operation 2:** Transform every occurrence of one existing character into another existing character, and do the same with the other character.
   For example, aacabb -> bbcbaa (all a's turn into b's, and all b's turn into a's)
3. You can use the operations on either string as many times as necessary.

However, your Android teacher isn't overly impressed, as the game can only be played with multiple participants, and in terms of complexity, it's relatively simple in terms of the number of screens. Eager to avoid losing marks, you request some leniency and negotiate. If you can develop an AI capable of playing this game, you'll receive extra credit. Though you regret the complexity of the rules, you'll have to make do.

Your task is to write a function that serves as an AI player, determining if the given words result in a "Bazinga." Return true if they do, and false otherwise.

Constraints:

- 1 <= word1.length, word2.length <= 10^5
- word1 and word2 contain only lowercase English letters.

Input Criteria

The first line contains an integer t, the number of test cases.
Each of the next two lines, the first line contains a string "word1" and the second line contains a string "word2"

Output Criteria

For each test case, print true if the words result in a "Bazinga" else false

Sample Input

3
abc
bca
a
aa
cabbba
abbccc

Sample Output

true
false
true

Problem B

ROBIN_HOOD_CASH

Tired from programming daily, you become sad and wander into the woods to find peace. You contemplate making money without coding but can't think of anything. Sadly, you realize you are only good at coding. Suddenly, you encounter a wizard who listens to you and understands your pain. He offers to help you by giving you some money... but in the form of coins. You argue that the money isn't enough, so he proposes another deal. He will give you magical sacks to put the coins in. You can take as many sacks as you like! These sacks are very special: whatever coins you put into them, after shaking them, the coins will all convert to the maximum coin value you put in. Wow!

However, the wizard gives you an ominous warning: if you put more than "k" coins into a sack, they will vanish! You cannot be too greedy because you are Robin Hood. Your goal is to decide on the number of sacks so that you can maximize your profit by putting at most "k" coins into each sack. This will help you achieve your dream of never having to work again and leaving PUCIT altogether to open a software house of your own, using coins.

Given a list of coins and the maximum capacity of each sack, find the maximum profit you could obtain by putting coins into the sacks sequentially.

For example, given a sack capacity of 3 and the coin values of 1, 15, 7, 9, 2, 5, and 10, you can divide them into 3 sacks so that the coins change to [15, 15, 15, 9, 10, 10, 10].

Constraints:

- 1 <= arr.length <= 500
- 0 <= arr[i] <= 10^9
- 1 <= k <= arr.length

Input Criteria

The first line contains an integer t, the number of test cases.
Each case consists of two lines, the first line contains an integer array of coins separated by a space and the second line contains the max sack capacity 'k'

Output Criteria

For each test case, print the highest profits obtained.

Sample Input

```
3
1 15 7 9 2 5 10
3
1 4 1 5 7 3 6 1 9 9 3
4
1
1
```

Sample Output

```
84
83
1
```

Problem C

Line Through Points

You're given an array of points plotted on a 2D graph (the xy-plane). Write a program that returns the maximum number of points that a single line (or potentially multiple lines) on the graph passes through.

The input array will contain points represented by an array of two integers [x, y]. The input array will never contain duplicate points and will always contain at least one point.

Input

First line will contain a number **n** that shows the number of points, Then there will be **n** lines and every line contains the position of that point(xi, yi).

Output

You have to return a single number that shows the maximum number of points that a single line on the graph passes through.

Example 1

Input	**Output**
7 1 1 2 2 3 3 0 4 -2 6 4 0 2 1	4

Example 2

Input	**Output**
3 1 4 4 1 3 3	2

Problem D

Jack and Jill

Jack stands at the foot of a staircase leading to Jill, eagerly awaiting him at the top of the hill. His mind, deeply absorbed in mathematics, calculates the many ways he can ascend to meet her. With the staircase boasting a total of n steps, Jack, despite his stature, can ascend one or two steps at a time or combine both in various sequences. His task is to discern the distinct methods available to reach Jill at the top, be it by starting with a single step and continuing or commencing with two steps and repeating the pattern. You want to discover how many different ways you can reach the top by mixing up, taking one step and two steps.

Consider the example with 4 steps, Jill can ascend the staircase in five possible ways: either by taking steps [1, 1, 1, 1], [1, 1, 2], [1, 2, 1], [2, 1, 1], or [2, 2].

Input
The input consists of a integer **t** number of test cases, followed by **t** lines of a integer **n** $(1 \leq n \leq 45)$

Output:
For each test case print an integer which shows the maximum possible ways.

Sample Input
5
1
2
3
4
5

Sample Output
1
2
3
5
8

Problem E

Crowded Cafe

Time limit per test: 1 second

Memory limit per test: 256 megabytes

As you and your friends rush into your favourite cafe one morning, you notice that most of the tables are reserved. The cafe layout comprises n rows of tables, with m tables in each row, forming an **n×m** grid. Each table can either be empty, represented by 'A', or reserved, represented by 'R'.

Your mission is to arrange **k** consecutive empty tables, in either the same row or column, to seat yourself and your friends together. You aim to explore how many different seating arrangements are possible under these conditions, ensuring that everyone sits together in a contiguous group. Two seating arrangements are distinct if the sets of occupied tables differ.

Let us calculate the number of ways you can arrange the tables to accommodate you and your friends comfortably, ensuring an enjoyable café experience for all!

Input

The first line contains three positive integers n, m, k ($1 \le n, m, k \le 2000$), where n, m represent the sizes of the cafe and k is the number of consecutive seats you need to find. Each of the next n lines contains m characters 'A' or 'R'. They form a matrix representing the café.

Output

A single number, denoting the number of ways to find k empty seats in the same row or column.

Examples 1

Input

```
2 3 2
RRA
AAA
```

Output

```
3
```

Examples 2

Input

```
1 2 2
AA
```

Output

```
1
```

Examples 3

Input

3 3 4
ARA
RAR
ARA

Output

0

Note

In the first sample, there are three ways to arrange those seats. You can take the following seats for your arrangement.

- (1,3), (2,3)
- (2,2), (2,3)
- (2,1), (2,2)