Question - 1
Casey's Image Editing

Casey has a square image made up of black and white pixels represented as *0* and *1* respectively. As part of an image analysis process, Casey needs to determine the size of the largest square area of white pixels. Given a *2*-dimensional square matrix that represents the image, write a function to determine the length of a side of the largest square area made up of white pixels.

For example, the $n \times n = 5 \times 5$ matrix of pixels is represented as *arr = [[1,1,1,1,1], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,1,1]]*. A diagram of the matrix is:

```
1 1 1 1 1
1 1 1 0 0
1 1 1 0 0
1 1 1 0 0
1 1 1 1 1
```

The largest square sub-matrix is 3×3 in size starting at position *(0, 0), (1, 0)* or *(2, 0)*. The expected return value is *3*.

Function Description
Complete the function *largestMatrix* in the editor below. The function must return width of the largest square sub-matrix of white pixels.

largestMatrix has the following parameter:
  arr[arr[0][0],...arr[n-1][n-1]]: a 2D array of integers

Constraints
- $1 \le n \le 500$
- *arr[i][j]* $\in$ *{0, 1} (0 denotes black pixel and 1 denotes white pixel)*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of rows.
The second line contains an integer, n, the number of columns.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains *n* space-separated integers that describe *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
3
3
1 1 1
1 1 0
1 0 1
```

Sample Output

```
2
```

Explanation
(0, 0) to *(1,1)* is the maximum square sub-matrix that contains all white pixels.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
3
0 1 1
1 1 0
1 0 1
```

Sample Output

```
1
```

Explanation
There is no square sub-matrix of size greater than *1* that contains all white pixels.

Question - 2
Efficient Janitor

The janitor at Hacker High School is insanely efficient. By the end of each day, all of the waste from the trash cans in the school has been shifted into plastic bags which can carry waste weighing between 1.01 pounds and 3.00 pounds. All of the plastic bags must be dumped into the trash cans outside the school. The janitor can carry at most 3.00 pounds at once. One trip is described as selecting a few bags which together don't weigh more than 3.00 pounds, dumping them in the outdoor trash can and returning to the school. The janitor wants to make minimum number of trips to the outdoor trash can. Given the number of plastic bags, *n,* and the weights of each bag, determine the minimum number of trips if the janitor selects bags in the optimal way.

For example, given *n = 6* plastic bags weighing *weight = [1.01, 1.99, 2.5, 1.5, 1.01],* the janitor can carry all of the trash out in *3* trips: *[1.01 + 1.99 , 2.5, 1.5 + 1.01].*

Function Description
Complete the function *efficientJanitor* in the editor below. The function must return a single integer that represents the minimum number of trips to be made.

efficientJanitor has the following parameter(s):
  weight[weight[0],...weight[n-1]]: an array of floating-point integers

Constraints

- $1 \le n \le 1000$
- $1.01 \le weight[i] \le 3.0$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *weight.*

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *weight[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
5
1.50
1.50
1.50
1.50
1.50
```

Sample Output

```
3
```

Explanation

In this case, the janitor will carry the first *2* plastic bags together, the *3rd* and *4th* together and the last one alone to dispose of all of the trash in *3* trips.

▼ Sample Case 1

Sample Input For Custom Testing

```
4
1.50
1.50
1.50
1.50
```

Sample Output

```
2
```

Explanation

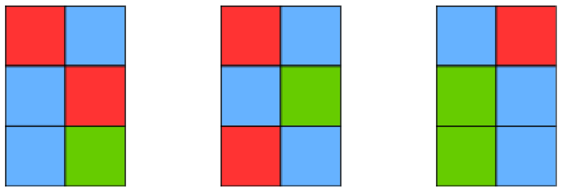
In this case, the janitor will carry the first *2* plastics bags together and the *3rd* and *4th* together requiring only *2* trips.

Question - 3
Improved Game of Twister

Max is developing a new version of the game of 3-D twister, where the cells on the mat have to be painted in three colors: Red, Blue, and Green. The mat is a 2-dimensional grid of size *3 x n.* Max wants to know how many different mats he can make, painting the cells in three colors, Red, Blue, and Green such that:

1. All the *n* cells of a single row don't have the same color.
2. The *3* cells of a single column are not all the same color.

For example, given *n=2,* there are *174* valid ways to paint the mat. Some of them are as shown in the image below:



Write code to compute the number of ways in which Max can paint the mat, using the given function.

Function Description

Complete the *twisterMat* function in the editor below. The function must return an integer denoting the number of ways in which you can paint the given grid. The result should be calculated as a mod of 10^9+7.

twisterMat has only one parameter:

n: An integer, denoting the number of columns of the mat.

Constraints

- $2 \le n \le 20000$

▼ Input Format For Custom Testing

The only line of input contains an integer, *n*, denoting the number of columns of the given mat.

▼ Sample Case 0

Sample Input For Custom Testing

```
2
```

Sample Output

```
174
```

Explanation

$n = 2$

Some valid ways to fill the mat:

1.
```
RedBlue
BlueRed
BlueGreen
```

2.
```
RedBlue
BlueGreen
RedBlue
```

3.
```
BlueRed
GreenBlue
GreenBlue
```

Some invalid ways are:

1.
```
RedBlue
RedGreen
RedBlue
```

2.
```
GreenGreen
GreenBlue
BlueBlue
```

▼ Sample Case 1

Sample Input For Custom Testing

```
3
```

Sample Output

```
9750
```

Explanation

n = 3

Some valid way to fill the grid:

1.
   ```
   BlueBlueGreen
   GreenRedRed
   RedRedBlue
   ```

2.
   ```
   BlueRedBlue
   GreenBlueGreen
   BlueRedRed
   ```

3.
   ```
   BlueRedBlue
   GreenBlueGreen
   GreenRedBlue
   ```

Some invalid ways are:

1.
   ```
   RedRedRed
   BlueRedGreen
   GreenRedBlue
   ```

2.
   ```
   BlueBlueBlue
   RedRedGreen
   GreenGreenRed
   ```

Question - 4
Strokes to paint

Alex wants to paint a picture but hates taking the brush off the canvas. In one stroke, Alex can only paint the same colored cells which are joined via some edge.

Given the painting, determine the minimum number of strokes to completely paint the picture.

Take for example, the canvas with height given by *h = 3* and width given by *w = 5* is to be painted with picture *picture=["aabba", "aabba", "aaaca"]*, the diagram below shows the *4* strokes needed to paint the canvas.

```
       Strokes
Canvas  1   2  3 4
aabba   aa  bb   a
aabba   aa  bb   a
aaaca   aaa    c a
```

Function Description

Complete the function *strokesRequired* in the editor below. The function must return an integer, the minimum number of strokes required to paint the canvas.

strokesRequired has the following parameter(s):

picture[picture[0],...picture[h-1]]: an array of strings where each string represents one row of the picture to be painted

Constraints

- *$1 \le h \le 10^5$*
- *$1 \le w \le 10^5$*
- *$1 \le h*w \le 10^5$*
- *len(picture[i]) = w (where $0 \le i < h$)*
- *picture[i][j] ∈ {'a', 'b', 'c'} (where $0 \le i < h$ and $0 \le j < w$)*

▼ Input Format For Custom Testing

The first line contains an integer, *h*, that denotes the height of the picture and the number of elements in *picture*.

Each line *i* of the *h* subsequent lines (where *$0 \le i < h$*) contains a string that describes *picture[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
3
aaaba
ababa
aaaca
```

Sample Output

```
5
```

Explanation

The 'a's can be painted in *2* strokes, 'b's in *2* strokes and 'c' in *1* stroke, for a total of 5.

```
        Strokes
 Canvas  1   2 3 4 5
 aaaba  aaa    b   a
 ababa  a a  b b   a
 aaaca  aaa      c a
```

▼ Sample Case 1

Sample Input For Custom Testing

```
4
bbba
abba
acaa
aaac
```

Sample Output

```
4
```

Explanation

The 'a's can be painted in *1* stroke, the 'b's in *1* stroke and each 'c' requires *1* stroke.

```
        Strokes
 Canvas  1   2   3 4
 bbba   bbb    a
 abba    bb a  a
 acaa       a aa c
 aaac       aaa    c
```

Question - 5
Angry Animals

Pi's father, Danny, runs the Hackerville Zoo. He is moving to Rookieville and wants to take all of the zoo animals with him via ship. He is confused about how to arrange them because a few of the species cannot be kept together in the same cabin.

There are *n* animals placed in a straight line. Each animal is identified by a unique number from *1* to *n* in order. There are *m* pairs *(a[i], b[i])* which imply that animals *a[i]* and *b[i]* are enemies and should not be kept in the same cabin. Pi is good at solving problems and he came up with following challenge: count the number of different groups that do not contain any pair such that they are enemies. A group is defined as an interval (x, y) such that all animals in the range from x to y form a group. Determine the number of groups that can be formed according to the Pi's challenge.

For example, given *n = 3* animals and *m = 3* pairs of enemies, *a = [1, 2, 3]* and *b = [3, 3, 1]*, animal *1* is the enemy of animal *3*, and animal *3* is the enemy of animals *1* and *2*. Because *3* is an enemy of both *1* and *2*, it must be in its own cabin. Animals *1* and *2* can be roomed together or separately. There are four possible groupings meeting the constraints: *{1, 2}* , *{1}*, *{2}*, *{3}*. Note that the intervals are along the original line of animals numbered consecutively from *1* to *n*, i.e. *[1, 2, 3]* in this case. They cannot be reordered.

Function Description
Complete the function *angryAnimals* in the editor below. The function must return the number of groups that can be formed according to Pi's challenge.

angryAnimals has the following parameters:
n: an integer that denotes the number of unique animals
a[a[0],...a[m-1]]: an array of integers
b[b[0],...b[m-1]]: an array of integers

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 10^6$
- $1 \le a[i], b[i] \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*.
The second line contains an integer, *m*, that denotes the number of elements in *a*.

Each line *i* of the *m* subsequent lines (where $0 \le i < m$) contains an integer that describes *a[i]*.
The next line again contains an integer, *m*, that denotes the number of elements in *b*.

Each line *i* of the *m* subsequent lines (where *0 ≤ i < m*) contains an integer that describes *b[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

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

Sample Output

```
7
```

Explanation
(1), (1,2), (2), (2,3), (3), (3,4), (4) are the groups that be formed according to Pi's challenge.

▼ Sample Case 1

Sample Input For Custom Testing

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

Sample Output

```
11
```

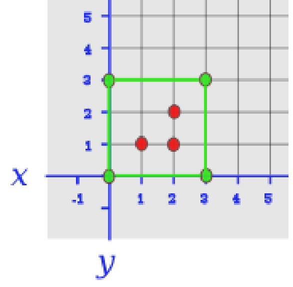
Explanation
(1), (1,2), (2), (2,3), (2,3,4), (3), (3,4), (3,4,5), (4), (4,5), (5) are the groups that can be formed according to Pi's challenge.

Question - 6
Minimum Area

Given a list of points described by their *(x,y)* coordinates on a two dimensional plane, construct a square surrounding at least a given number of points within the area enclosed. That area should be minimal and the square must meet the following conditions:

- The *x-coordinates* and *y-coordinates* of the points should be *integers*.
- The *sides* of the square should be *parallel to coordinate axes*.
- At least *k* of the given *n* points should *lie strictly inside the square drawn. Strictly inside means that they cannot lie on a side of the square.*

For example, given *n=3* points *(1,1), (1,2) and (2,1)* and *k=3*, surround all three points. The minimum area square is *9* units, going from the origin *(0,0)*, to *(3,3)*.



Function Description

Complete the function *minArea* in the editor below. The function must return the *minimum possible area* of the square that satisfies the constraints, as an integer.

minArea has the following parameter(s):
  x[x[0],...x[n-1]]: an array of integer *x* coordinates
  y[y[0],...y[n-1]]: an array of integer *y* coordinates
  k: an integer, the minimum number of points to surround

Constraints

- $2 \le n \le 100$
- $-10^9 \le x[i], y[i] \le 10^9$
- $1 \le k \le n$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *x*.
Each of the next *n* lines contains an integer *x[i]* where $0 \le i < n$.
The next line contains the integer *n*, the size of the array *y*.
Each of the next *n* lines contains an integer *y[i]* where $0 \le i < n$.
The last line contains the integer *k*.

▼ Sample Case 0

Sample Input 0

```
2
0
2
2
0
4
2
```

Sample Output 0

```
36
```

Explanation 0

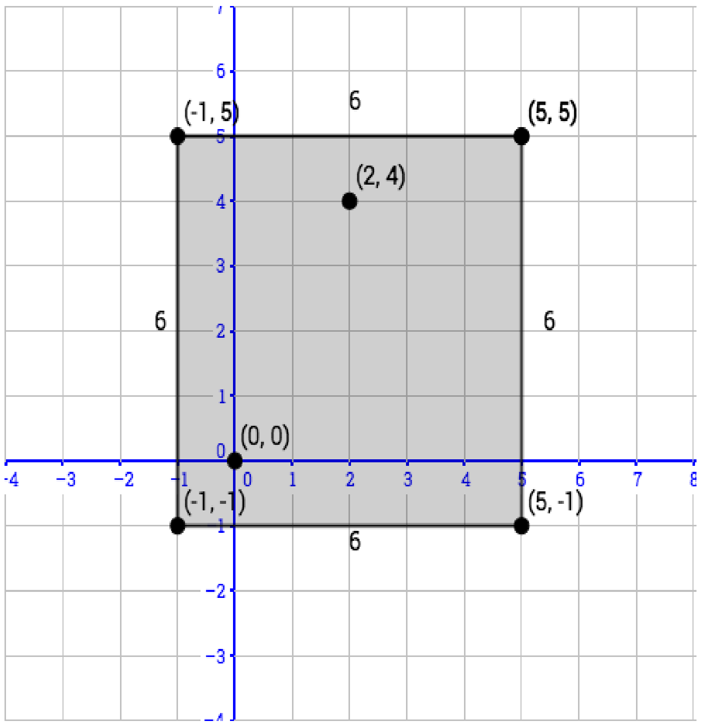
The given points are:

- *(0, 0)*
- *(2, 4)*

Choose following *four* points:

- *(-1, -1)*
- *(-1, 5)*
- *(5, 5)*
- *(5, -1)*

Draw a square of side *six*, satisfying the three constraints given in the problem statement and the area of the square is the minimum possible.



So, the function returns *36*, as the area of the square is *side x side (6 x 6 = 36)*.

▼ Sample Case 1

Sample Input 1

```
2
0
3
2
0
7
2
```

Sample Output 1

```
81
```

Explanation 1

The given points are:

- *(0, 0)*
- *(2, 7)*

Choose following *four* points:

- *(-1, -1)*
- *(-1, 8)*
- *(8, 8)*
- *(8, -1)*

Draw a square of side *nine* that satisfies the three constraints given in the problem statement and the area of the square is the minimum possible. The function returns *81 (9 x 9)*.

Question - 7
Arrange the Words

We define a *sentence* to be a string of space-separated words that starts with a capital letter followed by lowercase letters and spaces, and ends with a period, i.e., it satisfies the regular expression *^[A-Z][a-z]*\.$*. We want to rearrange the words in a *sentence* such that the following conditions are satisfied:

1. Each word is ordered by length, ascending.
2. Words of equal length must appear in the same order as in the original sentence.
3. The rearranged *sentence* must be formatted to satisfy the regular expression *^[A-Z][a-z]*\.$*

For example, consider the sentence *Cats and hats.* First the words are ordered by length, maintaining original order for ties: *[and, cats, hats]*. Now reassemble the sentence, applying formatting: *And cats hats.*

Function Description

Complete the function *arrange* in the editor below. The function must return a properly formed sentence arranged as described.

arrange has the following parameter(s):
sentence: a well formed sentence string

Constraints

- $1 \leq |sentence| < 10^5$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

A single line of space-separated words denoting *sentence*.

▼ Sample Case 0

Sample Input 0

```
The lines are printed in reverse order.
```

Sample Output 0

```
In the are lines order printed reverse.
```

Explanation 0

We organize the strings of each respective length in *sentence* = *The lines are printed in reverse order.* as:

- *Length 2: {in}*
- *Length 3: {the, are}*
- *Length 5: {lines, order}*
- *Length 7: {printed, reverse}*

We then reassemble our sequences of words as a sentence string, ensuring that the first letter is uppercase, the intermediate letters are lowercase, and the string terminates with a period. We return *In the are lines order printed reverse.* as our answer.

▼ Sample Case 1

Sample Input 1

```
Here i come.
```

Sample Output 1

```
I here come.
```

Explanation 1

We organize the strings of each respective length in *sentence = Here i come.* as:

- *Length 1: {i}*
- *Length 4: {here, come}*

We then reassemble and format our sentence as: *I here come.*.

▼ Sample Case 2

Sample Input 2

```
I love to code.
```

Sample Output 2

```
I to love code.
```

Explanation 2

We organize the strings of each respective length in *sentence = I love to code.* as:

- *Length 1: {i}*
- *Length 2: {to}*
- *Length 4: {love, code}*

We then reassemble and format our string as *I to love code.*.

Question - 8
Aladdin and his Carpet

Aladdin wants to travel around the world and will choose a circular path to fly on his magical carpet. The carpet needs enough magic to take him from one place to another. He knows that after traveling some distance, he can find a magic source that will enable the carpet to travel a further distance.

There are *n* magical sources along the circular path numbered from *0* to *n-1*. Initially, the carpet has no magic and Aladdin can use a portal to jump to any magical source and start his journey. The carpet consumes units of magic equal to the units of distance travelled. He needs to choose a point to start his journey that will allow him to complete his journey. Determine the lowest index of the starting points from which Aladdin can start his journey and visit all of the places in the circular path in order. If there is no solution, return -1.

For example, there are *n = 4* sources of magic along his route: *magic = [3, 2, 5, 4]* and *dist = [2, 3, 4, 2].* The first attempt is starting at the first source, *magic[0] = 3.* He transports there without cost and collects

3 units of magic. The distance to the next point is *dist[0] = 2.* It takes *2* units of magic to get there and he collects *magic[1] = 2* units upon arrival, so he has *3 - 2 + 2 = 3* units of magic after making his first carpet ride. Continuing along the journey:

- *3 - dist[1] + magic[2] = 3 - 3 + 5 = 5*
- *5 - dist[2] + magic[3] = 5 - 4 + 4 = 5*
- *5 - dist[3] = 5 - 2 = 3*

At this point, he is back to the first source. Because he can complete his journey starting at source *magic[0]*, there is no reason to continue with the analysis so its index, *0,* is returned. To illustrate a point from the same example, if he starts at position *2*, where *magic[1] = 2* and *dist[1] = 3*, he will not be able to proceed to the next point because the distance is greater than his magic units. Note that the list is circular, so from *magic[3]* in this example, the next source on the path is *magic[0].*

Function Description

Complete the function *optimalPoint* in the editor below. The function must return an integer that denotes the minimum index of *magic* from which he can start a successful journey. If no such starting point exists, return *-1*.

optimalPoint has the following parameter(s):

magic[magic[0],...magic[n-1]]: an array of integers where *magic[i]* denotes the amount of magic in the i^{th} source.

dist[dist[0],...dist[n-1]]: an array of integers where *dist[i]* denotes the distance to the next magical source.

Constraints

- $1 \le n \le 100000$
- $0 \le magic[i] \le 10000$
- $0 \le dist[i] \le 10000$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *magic*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *magic[i]*.

The next line again contains the integer, *n*, that denotes the number of elements in *dist*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *dist[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
4
2
4
5
2
4
4
3
1
3
```

Sample Output

```
1
```

Explanation

Here *magic = [2, 4, 5, 2]* and *dist = [4, 3, 1, 3].* If Aladdin starts at the second magical source, his magic levels are:

- *magic[1] = 4*
- *4 - dist[1] + magic[2] = 4 - 3 + 5 = 6*
- *6 - dist[2] + magic[3] = 6 - 1 + 2 = 7*
- *7 - dist[3] + magic[0] = 7 - 3 + 2 = 6*
- *6 - dist[0] = 6 - 4 = 2.*

The first point from where Aladdin can start his journey is the *2nd* magical source. The output should be *1,* the index of the *2nd* location.

▼ Sample Case 1

Sample Input For Custom Testing

```
4
8
4
1
9
4
10
9
3
5
```

Sample Output

```
-1
```

Explanation

Here *magic = [8, 4, 1, 9]* and *dist = [10, 9, 3, 5].* In each case, the distance to the next source is greater than the amount of magic at the current source. No matter where Aladdin starts, he will not be able to finish his travel.