Question - 1

SCORE: **50 points**

Shortest Substring

Algorithms | Strings | Data Structures | Problem Solving | Easy | Sets | HashMaps

Given a string comprised of lowercase letters in the range *ascii[a-z]*, determine the length of the smallest substring that contains all of the letters present in the string.

For example, given the string *s = dabbcabcd*, the list of all characters in the string is *[a, b, c, d]*. Two of the substrings that contain all letters are *dabbc* and *abcd*. The shortest substring containing all the letters is *4* characters long, *abcd*.

Function Description

Complete the function *shortestSubstring* in the editor below. The function must return the length of the shortest substring that contains all of the characters within *s*.

shortestSubstring has the following parameter:

s: a string

Constraints

- $1 \leq size\ of\ s \leq 10^5$
- $s[i] \in ascii['a'-'z']$

▼ Input Format For Custom Testing

The first line contains a string, *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
bab
```

Sample Output

```
2
```

Explanation

"ba" is a substring that contains all the characters in *s*.

▼ Sample Case 1

Sample Input For Custom Testing

```
bcaacbc
```

Sample Output

```
3
```

Explanation

"bca" is a substring that contains all the characters in *s*.

Question - 2

SCORE: **75 points**

List Max

Data Structures | Medium | Algorithms | Arrays | Core Skills | Problem Solving

In this challenge, you will start with an array initialized to zeros with indexes starting at 1. You will then be given a series of operations to perform on segments of the list. Each operation will consist of a starting and ending index within the array, and a number to add to each element within that range.

For example, start with an array of 5 elements: *[0, 0, 0, 0, 0].* The variables *a* and *b* represent the starting and ending indexes inclusive. Another variable *k* is the addend. The first element is at index 1.

```
a    b    k              list
                  [  0,  0,  0,  0,  0]
1    2   10     [ 10, 10,  0,  0,  0]
2    4    5     [ 10, 15,  5,  5,  0]
3    5   12     [ 10, 15, 17, 17, 12]
```

The maximum value in the resultant array is 17. That is the value to be determined.

Function description

Complete the *listMax* function in the editor below. The function must return a long integer denoting the largest value of the array after all operations have been performed.

listMax has the following parameters:

n: An integer, denoting the size of the array.

operations: A 2D integer array, denoting the operations.

Constraints

- $3 \le n \le 10^7$
- $1 \le o \le 2 \times 10^5$
- $1 \le a \le b \le n$
- $0 \le k \le 10^9$

▼ Input Format

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

The first line contains an integer, *n*, the size of your array

The second line contains an integer, *o*, the number of operations

The next line contains the integer 3, the number of elements used to define each operation

o lines follow, each containing 3 space-separated integers, *a*, *b*, and *k*: the starting index, ending index and value to add

▼ Sample Case 0

Sample Input 0

```
5
3
3
1 2 100
2 5 100
3 4 100
```

Sample Output 0

```
200
```

Explanation 0

We perform the following sequence of *o = 3* operations on *list = [0, 0, 0, 0, 0]*:

1. Add *k = 100* to every element in the inclusive range *[1, 2]*, resulting in *list = [100, 100, 0, 0, 0]*.
2. Add *k = 100* to every element in the inclusive range *[2, 5]*, resulting in *list = [100, 200, 100, 100, 100]*.
3. Add *k = 100* to every element in the inclusive range *[3, 4]*, resulting in *list = [100, 200, 200, 200, 100]*.

We then print the maximum value in the final list, *200*, as our answer.

▼ Sample Case 1

Sample Input 1

```
4
3
3
2 3 603
1 1 286
4 4 882
```

Sample Output 1

```
882
```

Explanation 1

We perform the following sequence of *o = 3* operations on *list = [0, 0, 0, 0]*:

1. Add *k = 603* to every element in the inclusive range *[2, 3]*, resulting in *list = [0, 603, 603, 0]*.
2. Add *k = 286* to every element in the inclusive range *[1, 1]*, resulting in *list = [286, 603, 603, 0]*.
3. Add *k = 882* to every element in the inclusive range *[4, 4]*, resulting in *list = [286, 603, 603, 882]*.

We then print the maximum value in the final list, *882*, as our answer.

Question - 3

SCORE: **75 points**

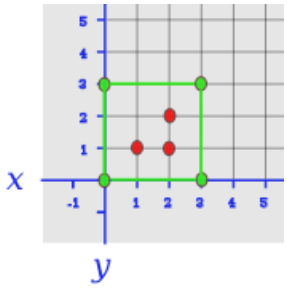
Minimum Area

Greedy | Medium | Algorithms | Core Skills | Problem Solving

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 satisfying 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

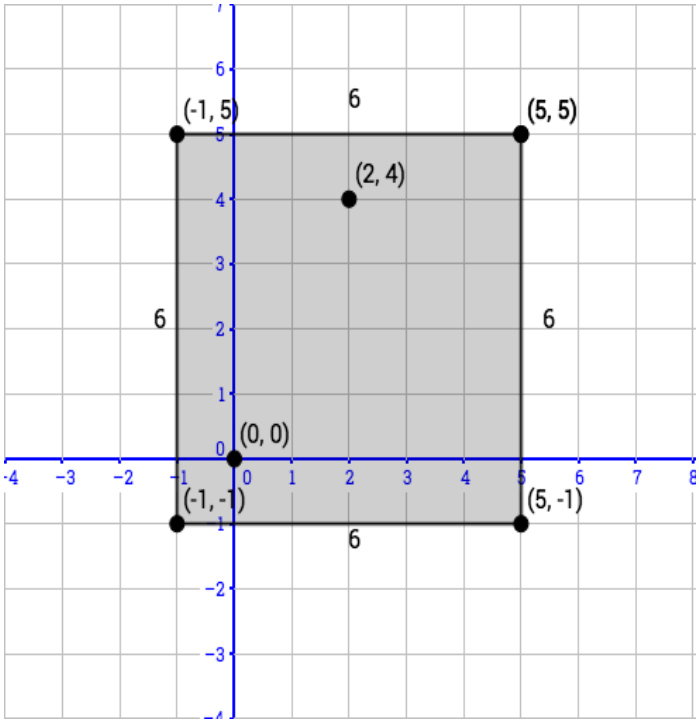
The given points are:

- *(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)*.