**1.Java: Real Estate Listing Management**

A customer requested a real estate application to manage their real estate listings. They want to have the ability to add, remove, update, and retrieve listings based on specific criteria.

Create a new class called *RealEstateListing* and implement the *IRealEstateListing* interface.

* Add the following properties to the *RealEstateListing* class:
  + *ID* (int)
  + *title* (String)
  + *description* (String)
  + *price* (int)
  + *location* (String)

Create another class called *RealEstateApp* and implement the *IRealEstateApp* interface.

* Declare a private field within the *RealEstateApp* class called *listings*, which is a *List<IRealEstateListing>*.
* Implement the following methods within the *RealEstateApp* class:
  + *addListing(IRealEstateListing):* add the listing to *listings*
  + *removeListing(int listingID):* if a listing with this *ID* exists, remove it from *listings*
  + *updateListing(IRealEstateListing):* update the listing with a matching *ID* to the values in the parameter object
  + *getListings():* return *listings*
  + *getListingsByLocation(String location):* return a new list of the listings that match the *location* parameter
  + *getListingsByPriceRange(int minPrice, int maxPrice):* return a new list of the listings whose price falls within the specified range, inclusive

**Example**

There are 2 RealEstateListing objects, with Id, Title, Description, Price, Location.

*1 Serene-Haven Spacious-3-bedroom-home 209 Green-Valley*

*2 Luxury-Oasis Elegant-4-bedroom-villa-with-a-private-pool 440 Riverside-City*

Filter by this location.

*Riverside-City*

Filter for this price range.

*182 399*

**Output:**

All Listings:

ID: 1, Title: Serene-Haven, Price: 209 , Location: Green-Valley

ID: 2, Title: Luxury-Oasis, Price: 440 , Location: Riverside-City

Listings in Riverside-City:

ID: 2, Title: Luxury-Oasis, Price: 440

Listings By Price Range (182 - 399):

ID: 1, Title: Serene-Haven, Price: 209

Input Format For Custom Testing

The first line contains an integer *n*, the number of *RealEstateListing* objects.

Each of the next *n* lines contains a *RealEstateListing* object's information as 5 space-separated values: *ID Title Description Price Location*.

The next line contains the string, the *location* to search.

The next line contains two space-separated integers, *minPrice* and *maxPrice*.

Sample Case 1

**Sample Input For Custom Testing**

STDIN                                    Function

-----                                    --------

8                                        number of listings n = 8

1 Listing-1 Description-1 195 Location-0 first listing: Id = 1, Title = Listing-1, Description = 'Description-1', Price = 195, Location = 'Location-0'

2 Listing-2 Description-2 177 Location-1 second listing ...

3 Listing-3 Description-3 337 Location-3

4 Listing-4 Description-4 196 Location-2

5 Listing-5 Description-5 239 Location-3

6 Listing-6 Description-6 477 Location-0

7 Listing-7 Description-7 120 Location-0

8 Listing-8 Description-8 219 Location-1

Location-1                               *location = 'Location-1'*

357 407                                 minPrice = 357, maxPrice = 487

**Sample Output**

All Listings:

ID: 1, Title: Listing-1, Price: 195 , Location: Location-0

ID: 2, Title: Listing-2, Price: 177 , Location: Location-1

ID: 3, Title: Listing-3, Price: 337 , Location: Location-3

ID: 4, Title: Listing-4, Price: 196 , Location: Location-2

ID: 5, Title: Listing-5, Price: 239 , Location: Location-3

ID: 6, Title: Listing-6, Price: 477 , Location: Location-0

ID: 7, Title: Listing-7, Price: 120 , Location: Location-0

ID: 8, Title: Listing-8, Price: 219 , Location: Location-1

Listings in Location-1:

ID: 2, Title: Listing-2, Price: 177

ID: 8, Title: Listing-8, Price: 219

Listings By Price Range (357 - 407):

**Explanation**

There are 8 listings to add. The code stub reads the data, makes the method calls, and generates results.

Sample Case 2

**Sample Input For Custom Testing**

1

1 Listing-1 Description-1 289 Location-3

Location-0

259 472

**Sample Output**

All Listings:

ID: 1, Title: Listing-1, Price: 289 , Location: Location-3

Listings in Location-0:

Listings By Price Range (259 - 472):

ID: 1, Title: Listing-1, Price: 289

**Explanation**

There is 1 listing to add. The code stub reads the data, makes the method calls, and generates results.

**PROBLEM SOLVING**

**1.Academic Decathlon**

In a school, there are *n* students who want to participate in an academic decathlon. The teacher wants to select the maximum number of students possible. Each student has a certain skill level. For the team to be uniform, it is important that when the skill levels of its members are arranged in increasing order, the difference between any two consecutive skill levels is either *0* or *1.*Find the maximum team size the teacher can form.

**Example**

*skills = [10, 12, 13, 9, 14]*

Valid teams, sorted are *{9, 10},*and *{12, 13, 14}.*These teams have team sizes *2*and *3* respectively, so the maximum team size is *3.*

**Function Description**

Complete the function *findMaxTeamSize* in the editor below.

*findMaxTeamSize* has the following parameter:

*int skills[n]:*  the skill levels of each student

**Returns**

*int:*  the maximum possible size of the team

**Constraints**

* *1 ≤ n ≤ 105*
* *1 ≤ skills[i] ≤ 109*

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *skills*.  
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *skills[i]*.

Sample Case 0

**Sample Input 0**

STDIN    Function

-----    --------

4    →    skills[] size n = 4

4    →    skills = [4, 13, 2, 3]

13

2

3

**Sample Output 0**

3

**Explanation 0**

There are two valid teams possible: *{2, 3, 4}* and *{13}*. These have team sizes of *3* and *1*, respectively.

**2.Backspace String Compare**

Two strings are said to be the same if they are of the same length and have the same character at each index. Backspacing in a string removes the previous character in the string.

Given two strings containing lowercase English letters and the character **'#'** which represents a backspace key, determine if the two final strings are equal. Return 1 if they are equal or 0 if they are not. Note that backspacing an empty string results in an empty string.

**Example**

*s1 = 'axx#bb#c'*

*s2 = 'axbd#c#c'*

In the first string, one 'x' and one 'b' are backspaced over. The first string becomes *axbc. T*he second string also becomes *axbc.* The answer is 1.

**Function Description**

Complete the function *compareStrings* in the editor below.

compareStrings has the following parameter(s):

*string s1:*the first string

*string s2:* the second string

**Returns**

***int:* either 0 or 1**

**Constraints**

* 1 ≤  length of s1 ≤ 2\*105
* 1 ≤  length of s2 ≤ 2\*105
* Both s1 and s2 contain lowercase English letters and/or the character '#' only.

Input Format For Custom Testing

The first line contains a string, *s1*. The second line contains a string, *s2*.

Sample Case 0

**Sample Input For Custom Testing**

yf#c#

yy#k#pp##

**Sample Output**

1

**Explanation**

Both the strings s1 and s2 result in "y" after processing backspaces.

Sample Case 1

**Sample Input For Custom Testing**

hacc#kk#

hb##ackk##

**Sample Output**

0

**Explanation**

The first string becomes "hack" while the second string becomes "ac".

**3.Beautiful String**

A string is *beautiful* if no two adjacent characters are either

1. the same, for example 'aa'.
2. adjacent in the alphabet, for example 'ef'.

The following operations can be performed on a string, *s.*

* Choose any index *i*(0 ≤ *i*< |s|) and change *s[i]* to any lowercase English letter.

Find the minimum number of operations required to make the string beautiful.

Example

*s =* "abdde"

String *s* is not beautiful because:

* 'dd' violates constraint 1, no two adjacent characters are the same.
* 'ab' and 'de' violate constraint 2, no two adjacent characters are adjacent in the alphabet.

The string can be converted into a beautiful string after 2 operations. One solution is below.

* Choose *i*=1 and change *s[i]* to 'z'.*s* becomes "azdde".
* Choose *i*=3 and change *s[i]* to 'k'.*s* becomes "azdke" which is beautiful.

Note: There are many other solutions such as "ardze", "axdke", etc.

It can be shown that 2 is the minimum number of operations required so return 2.

Function Description

Complete the function *getMinimumOperationCount* in the editor below.

*getMinimumOperationCount* has the following parameter:

*s:*  a string

Returns

*int:* the minimum number of operations required to make *s* beautiful

Constraints

* 2 ≤ |*s*| ≤ 105
* The string *s* contains only lowercase English letters.

Input Format For Custom Testing

The first line contains a string, *s*.

Sample Case 0

Sample Input For Custom Testing

STDIN        FUNCTION

-----        --------

bcbb    →    s = "bcbb"

Sample Output

2

Explanation

"bcbb" contains adjacent characters which are adjacent in the alphabet, i.e. "cb", "bc", and adjacent matching characters "bb".

One alternative is:

* Choose *i*=1, change *s[i]* to 'd'.*s* becomes "bdbb".
* Choose *i*=3, change *s[i]* to 'd'.*s* becomes "bdbd" which is beautiful.

Sample Case 1

Sample Input For Custom Testing

STDIN        FUNCTION

-----        --------

aceb    →    s = "aceb"

Sample Output

0

Explanation

String *s* is already beautiful.

**4.Compression**

Different compression techniques are used in order to reduce the size of the messages sent over the web. An algorithm is designed to compress a given string by describing the total number of consecutive occurrences of each character next to it. For example, consider the string *"abaasass".* Group the consecutive occurrence of each character:

* *'a'* occurs one time.
* *'b'* occurs one time.
* *'a'* occurs two times consecutively.
* *'s'* occurs one time.
* *'a'* occurs one time.
* *'s'* occurs two times consecutively.

If a character only occurs once, it is added to the compressed string. If it occurs consecutive times, the character is added to the string followed by an integer representing the number of consecutive occurrences. Thus the compressed form of the string is *"aba2sas2"*.

**Function Description**

Complete the function *compressedString* in the editor below. The function must return the compressed form of *message*.

compressedString has the following parameter(s):

*string message:*  a string

Returns:

*string:* the compressed message

**Constraints**

* *message[i] ∈ ascii[a-z]*
* *|message| ≤ 105*

Input Format For Custom Testing

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

The only line of the input contains the string *message*.

Sample Case 0

**Sample Input 0**

STDIN    Function Parameters

-----    -------------------

abc →  message = "abc"

**Sample Output 0**

abc

**Explanation 0**

None of the characters repeats consecutively so the string is already in compressed form.

Sample Case 1

**Sample Input 1**

STDIN        Function Parameters

-----        -------------------

abaabbbc →  message = "abaabbbc"

**Sample Output 1**

aba2b3c

**Explanation 1**

Group the consecutive occurrences of each character to get *"{a}{b}{aa}{bbb}{c}"*, in compressed form: *"aba2b3c"*.

**5.Count Balanced Subarray**

There are *n* software components, each with an integer value represented by *componentValue[i]*, where *0 ≤ i < n*.

Two teams manage these components:

* One team handles components with odd values.
* The other manages components with even values.

A subarray is considered *balanced* if it contains:

* An odd number of odd-valued components.
* An even number of even-valued components.

Count the number of balanced subarrays in *componentValue*.

**Example**

*n* = 3

*componentValue* = [2, 4, 3]

Balanced subarrays:

* [3] (1 odd, 0 even)
* [2, 4, 3] (1 odd, 2 even)

Non-balanced subarrays include:

* [2, 4] (0 odd, 2 even)
* [4, 3] (1 odd, 1 even)
* [2] (0 odd, 1 even)

Return the number of balanced subarrays, 2.

**Function Description**

Complete the function *countBalancedSubarrays* in the editor below.

**Function Parameters**

*int componentValue[n]*: the component values

**Returns**

*long:* the number of balanced subarrays in *componentValue*.

**Constraints**

* 1 ≤*n* ≤ 2 \* 105
* 1 ≤ *componentValue[i]*≤ 109

**Input Format for Custom Testing**

The first line contains an integer *n*, the size of *componentValue.*

Each of the next n lines contains an integer, *componentValue[i].*

**Sample Case 0**

**Sample Input 0**

STDIN     Function

-----     --------

3     →   componentValue[] size n = 3

1     →   componentValue = [1, 2, 3]

2

3

**Sample Output 0**

2

**Explanation**

The only balanced subarrays are [1], and [3].

**Sample Case 1**

**Sample Input 1**

STDIN     Function

-----     --------

4     →   componentValue[] size n = 4

1     →   componentValue = [1, 4, 3, 2]

4

3

2

**Sample Output 1**

3

**Explanation**

The balanced subarrays are [1], [3], [4, 3, 2].

**6.Counting Triplets**

There is an integer array *arr[n]* and an integer value *d.* The array is indexed from *1* to*n.*

Count the number of distinct triplets *(i, j, k)* such that *0 < i < j < k ≤ n* and the sum *(a[i] + a[j] + a[k])* is divisible by *d.*

**Example**

*a* = *[3, 3, 4, 7, 8]*

*d = 5.*

The following triplets are divisible by *d = 5.*  Following are the triplets whose sum is divisible by *d (1-based indexing):*

* indices (1, 2, 3), sum = 3+3+4 = 10
* indices (1, 3, 5), sum = 3+4+8 = 15
* indices (2, 3, 4), sum = 3+4+8 = 15

Since there is no other triplet divisible by *d = 5*, return *3.*

**Function Description**

Complete the function *getTripletCount* in the editor below.

*getTripletCount* has the following parameters:

*int a[n]:*  an array of integers

*int d:* an integer

**Returns**

*int:* the number of distinct triplets

**Constraints**

* *3 ≤ n ≤ 103*
* *1 ≤ a[i] ≤ 10 9*
* *2 ≤ d ≤ 106*

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in the array *a*.

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

The last line contains a single integer, *d*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN        FUNCTION

-----        --------

4       →    n = 4

2       →    a = [2, 3, 1, 6]

3

1

6

3       →    d = 3

**Sample Output**

2

**Explanation**

* indices  (1, 2, 3), sum = 2+3+1 = 6
* indices (1, 3, 4), sum = 2+1+6 = 9

Sample Case 1

**Sample Input For Custom Testing**

STDIN        FUNCTION

-----        --------

5       →    n = 5

5       →    a = [5, 3, 5, 5, 8]

3

5

5

8

11      →    d = 11

**Sample Output**

0

**Explanation**

There is no triplet whose sum is divisible by 11.

**7.Data Reorganization**

While analyzing data, you are working with an array *data* containing *n* positive integers, each representing dataset values.

To derive new features for data analysis, you can perform the following operation:

1. Select a pair of indices *(i, j)* (0-based) such that *0 ≤ i < j < len(data)*.
2. Compute the absolute difference *|data[i] - data[j]|*.
3. Append this value to the end of the array *data*, increasing its length by 1.

The objective is to minimize the smallest value present in *data* after performing exactly *maxOperations* operations.

Write a program to compute the minimum possible value of the smallest element in *data* after the given operations.

**Example**

*n* = 6

*data* = [42, 47, 50, 54, 62, 79]

*maxOperations* = 2.

The underlined values are selected for the operation.

| An optimal sequence of operation is as follows | | |
| --- | --- | --- |
| **Operation Number** | ***data[]* before** | ***data[]* after** |
| 1 | [42, 47, 50, 54, 62, 79] | [42, 47, 50, 54, 62, 79, 15] |
| 2 | [42, 47, 50, 54, 62, 79, 15] | [42, 47, 50, 54, 62, 79, 15, 3] |

The smallest possible value of the minimum element is 3.

**Function Description**

Complete the function *getMinimumValue* in the editor with the following parameters:

*int data[n]:* the dataset

*int maxOperations:* the number of operations to be performed

**Returns**

*int:* the smallest possible value of the minimum element in *data* after exactly *maxOperations* operations

**Constraints**

* 2 ≤ *n* ≤ 2\*103
* 1 ≤ *data[i]* ≤ 109
* 1 ≤ *maxOperations* ≤ 109

Input Format for Custom Testing

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

Each of the next *n* lines contains an integer *data[i]*.

The next line contains an integer *maxOperations*.

Sample Case 0

**Sample Input 0**

STDIN           Function

-----          --------

5         →     data[] size n = 5

4         →     data = [4, 2, 5, 9, 3]

2

5

9

3

1         →     maxOperations = 1

**Sample Output 0**

1

**Explanation**

The underlined values are selected for the operation.

| An optimal sequence of operation is as follows | | |
| --- | --- | --- |
| **Operation Number** | ***data[]* before** | ***data[]* after** |
| 1 | [4, 2, 5, 9, 3] | [4, 2, 5, 9, 3, 1] |

The smallest possible value of the minimum element is 1.

Sample Case 1

**Sample Input 1**

STDIN           Function

-----           --------

5         →     data[] size n = 5

5         →     data = [5, 18, 3, 12, 11]

18

3

12

11

2         →     maxOperations = 2

**Sample Output 1**

1

**Explanation**

The underlined values are selected for the operation.

| An optimal sequence of operation is as follows | | |
| --- | --- | --- |
| **Operation Number** | ***data[]* before** | ***data[]* after** |
| 1 | [5, 18, 3, 12, 11] | [5, 18, 3, 12, 11, 6] |
| 2 | [5, 18, 3, 12, 11, 6] | [5, 18, 3, 12, 11, 6, 1] |

The smallest possible value of the minimum element is 1.

**8.Data Transformation**

Given a dataset of strings containing only parentheses, characters ‘(‘ and ‘)’, the data represented by the string is valid if it is a balanced bracket sequence. One adjustment to the string can be made: at most one bracket can be moved from its original place to any other position in the string. The task is to determine whether, for each string, it is possible to balanced the bracket sequence in 1 move or less.  Return an array of the size of the dataset, where the *ith* integer is 1 if the string can be converted into a balanced string, and 0 otherwise.

Note: A string *s* is a balanced bracket sequence if:

* *s* is empty.
* *s* is equal to *"(t)"*, where *t* is a balanced bracket sequence.
* *s* is equal to *t1t2*, i.e. concatenation of*t1* and*t2*, where *t1* and *t2* are balanced bracket sequences.

**Example**

*n* = 3

*dataset*= [“)(“, “(()”, “()”],

For the first string ")(", applying the operation to move the first bracket to the end results in "()", which is a balanced bracket sequence.

For the second string "(()", it is impossible to convert it into a balanced bracket sequence.

For the third string "()", it is already a balanced bracket sequence.

Hence, the answer is [1, 0, 1].

**Function Description**

Complete the function *isConvertibleData* in the editor below.

*isConvertibleData* takes the following parameter(s):

*string dataset[n]:* the dataset where each string contains characters ')' and '('

**Returns**

*int[n]:* an array of integers, where the *ith* integer is 1 if the corresponding string can be transformed into a balanced bracket sequence and 0 otherwise

**Constraints**

* 1 ≤ *n* ≤ 2 \* 105
* 1 ≤ |*dataset[i]*|≤ 2 \* 105
* *n* ≤ Σ|*dataset[i]*| ≤ 2 \* 105
* It is guaranteed that each string *dataset[i]* consists of characters '(' and ')' only

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *dataset.*

Each of the next *n* lines contains a string *dataset[i].*

Sample Case 0

**Sample Input For Custom Testing**

STDIN FUNCTION

-----          --------

2 → dataset[] size n = 2

( → dataset = ["(", ")((())"]

)((())

**Sample Output**

0

1

**Explanation**

For *dataset[0]*= "(", it is not possible to convert the string to a balanced bracket sequence.

For *dataset[1]* = ")((())", the first bracket can be moved to the last position to obtain "((()))", which is balanced.

Sample Case 1

**Sample Input For Custom Testing**

STDIN FUNCTION

-----          --------

2 → dataset[] size n = 2

() → dataset = ["()", "))(("]

))((

**Sample Output**

1

0

**Explanation**

*dataset[0]* = "()", which is already a balanced bracket sequence.

For *dataset[1]*= "))((", it cannot be converted to a balanced bracket sequence in at most one move.

**9.Even Difference**

Consider every subsequence of an array of integers.

* Sort the subsequence in increasing order.
* Determine the sum of differences of elements in the subsequence.
* Return the length of the longest subsequence where this sum is even.

**Example**  
Given *n = 4* elements and *arr = [2, 4, 1, 7]*, these are some of the subsequences.

| **Subsequence** | **Sorted Subsequence** | Sum of diff of  Adjacent elements | **Is Valid** | **Length** |
| --- | --- | --- | --- | --- |
| [2, 4, 1] | [1, 2, 4] | 1 + 2 = 3 (Odd) | No | 3 |
| [2, 1, 7] | [1, 2, 7] | 1 + 5 = 6 (Even) | Yes | 3 |
| [2, 4, 1, 7] | [1, 2, 4, 7] | 1 + 2 + 3 = 6 (Even) | Yes | 4 |
| [2, 1] | [1, 2] | 1 (Odd) | No | 2 |

 We can see that the maximum possible length of a valid subsequence is 4.

**Function Description**

Complete the function *findLongestSubsequence* in the editor below.

*findLongestSubsequence* has the following parameter(s):

*int arr[n]:*  an array of integers

**Returns**

*int:* the length of the longest subsequence as described

**Constraints**

* 3 ≤ *n* ≤ 105
* 0 ≤ *arr[i]* ≤ 109

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer,*arr[i]*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN     FUNCTION

-----     ----------

7     → arr size [] n = 7

7     →  arr = [7, 5, 6, 2, 3, 2, 4]

5

6

2

3

2

4

**Sample Output**

6

**Explanation**

Consider the subsequence [5, 6, 2, 3, 2, 4].

* arrange the subsequence in ascending order, 2, 2, 3, 4, 5, 6
* the differences are 0,1,1,1,1
* 0+1+1+1+1 = 4

Sample Case 1

**Sample Input For Custom Testing**

STDIN     FUNCTION

-----     ----------

4     → arr size[] *n* = 4

1     →  arr= [1, 3, 5, 7]

3

5

7

**Sample Output**

4

**Explanation**

The entire array can be used.

* arrange the subsequence in ascending order, 1,3,5,7
* the adjacent differences are 2, 2, 2
* 2+2+2 = 6

**10.Extraordinary Substrings**

Each character of the lowercase English alphabet has been mapped to digits as shown in the figure. The numerical value corresponding to each letter is its mapped value.



An *extraordinary* substring is one whose sum of the mapped values of each letter is divisible by its length. Given string *input\_str*, count its total number of non-empty extraordinary substrings.

**Example:**

*input\_str = 'asdf'*

All non-empty substrings of *input\_str* are tested in the table.

String  Mapped  Sum   Length    Is divisible

a       1         1     1           Yes

s       7         7     1           Yes

d       2         2     1           Yes

f       3         3     1           Yes

as      1,7       8     2           Yes

sd      7,2       9     2           No

df      2,3       5     2           No

asd     1,7,2    10     3           No

sdf     7,2,3    12     3           Yes

asdf    1,7,2,3  13     4           No

There are 6 extraordinary substrings.

**Function Description**

Complete the function *countSubstrings* in the editor.

*countSubstrings* has the following parameter(s):

*string input\_str:* a string of length *n*

**Returns**

*int*: the number of non-empty extraordinary substrings

**Constraints**

* *1 ≤ n ≤ 2000*
* All characters of *input\_str* are lowercase English letters.

Input Format For Custom Testing

The first line contains a string, *input\_str.*

Sample Case 0

**Sample Input For Custom Testing**

STDIN    FUNCTION

-----   --------

bdh   → input\_str = "bdh"

**Sample Output**

4

**Explanation**

The extraordinary substrings are 'b', 'd', 'h' and 'bdh'.

Sample Case 1

**Sample Input For Custom Testing**

STDIN    FUNCTION

-----    --------

abcd → input\_str = "abcd"

**Sample Output**

6

**Explanation**

The extraordinary substrings are: 'a', 'b', 'c', 'd', 'ab' and 'cd'.

**11.1Find the Sequence Sum**

Given three integers, *i*, *j*, and *k*, a *sequence sum* to be the value of *i + (i + 1) + (i + 2) + (i + 3) + … + j + (j − 1) + (j − 2) + (j − 3) + … + k* (increment from *i* until it equals *j,* then decrement from *j* until it equals *k).* Given values *i, j,* and *k*, calculate the sequence sum as described.

Example

*i = 5*

*j = 9*

*k = 6*

Sum all the values from *i* to *j* and back to *k: 5 + 6 + 7 + 8 + 9 + 8 + 7 + 6 = 56*.

Function Description

Complete the function *getSequenceSum* in the editor below.

getSequenceSum has the following parameter(s):

*int i, int j, int k:*  three integers

Return

*long:* the value of the sequence sum

Constraints

* *-108 ≤ i, j, k ≤ 108*
* *i, k ≤ j*

Input Format For Custom Testing

The first line contains an integer, *i*.

The next line contains an integer, *j*.

The last line contains an integer, *k*.

Sample Case 0

Sample Input 0

STDIN    Function

-----    --------

0    → i = 0

5    →   j = 5

-1   →   k = -1

Sample Output 0

24

Explanation 0

*i = 0*

*j = 5*

*k = -1*

*0 + 1 + 2 + 3 + 4 + 5 + 4 + 3 + 2 + 1 + 0 + -1 = 24*

Sample Case 1

Sample Input 1

STDIN     Function

-----     --------

-5    →    i = -5

-1    →    j = -1

-3    →    k = -3

Sample Output 1

-20

Explanation 1

*i = -5*

*j = -1*

*k = -3*

*-5 + -4 + -3 + -2 + -1 + -2 + -3 = -20*

**112.Longest Even Length Word**

Consider a string, *sentence*, of *words* separated by spaces where each word is a substring that consists of English alphabetic letters only. Find and return the first word in the *sentence* that has a length that is both an even number and has the greatest length of all even-length words in the *sentence*. If there are no even length words in the sentence, return '00'.

Example

*sentence = "Time to write great code"*

The lengths of the words are 4, 2, 5, 5, 4, in order. The longest even length words are *Time* and *code*. The one that occurs first is *Time*, the answer to return.

Function Description

Complete the function *longestEvenWord* in the editor below.

longestEvenWord has the following parameter(s):

*string sentence:*  a sentence string

Returns:

*string:* the first occurrence of a string with maximal even number length, or the string '00' (zero zero) if there are no even length words

Constraints

* *1 ≤ length of sentence ≤ 105*
* The *sentence* string consists of spaces and letters in the range ascii[a-z, A-Z, ] only.

Input Format for Custom Testing

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

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

Sample Case 0

Sample Input 0

STDIN                                Function

--------------------------           -------------------------------

It is a pleasant day today    →      sentence = "It is a pleasant day today"

Sample Output 0

pleasant

Explanation 0

There are three even-length words: *It* (with length *2*), *is* (*2*), and *pleasant* (*8*).

Sample Case 1

Sample Input 1

STDIN                                    Function

------------------------------           -------------------------------

You can do it the way you like    →     sentence = "You can do it the way you like"

Sample Output 1

like

Explanation 1

There are three words of even length: *do* (with length *2*), *it* (*2*), and *like* (*4*).

**13.Maximal Substring**

A binary string *dataStream,* has an even length and consists of characters*'0'*and*'1'*. The following operation can be performed on the string any number of times:

Using 0-based indexing, choose two distinct even indices *i* and *j*and perform the following swaps:

1. Swap *dataStream[i]* with *dataStream[j].*
2. Swap *dataStream[i+1]* with *dataStream[j+1]*.

Find the length of the longest non-decreasing substring in *dataStream* after performing the swaps.

**Example**

*dataStream* = "110000"

A possible order of operations is:

* Choose *i* = 0 and *j* = 2, yielding "001100".
* Choose *i* = 0 and *j* = 4, yielding "001100".
* Choose *i* = 2 and *j* = 4, yielding "000011".

Hence, the maximum length of a non-decreasing substring is 6.

**Function Description**

Complete the function *getMaxSubstring* in the editor with the followingparameter:

*string dataStream:* the binary string

**Returns**

*int:* the maximum length of the non-decreasing substring possible

**Constraints**

* 2 ≤ *length of dataStream* ≤ 2 \* 105
* *dataStream[i]* is '0' or '1'
* The length of *dataStream* is even.

Input Format for Custom Testing

The first line contains the string *dataStream.*

Sample Case 0

**Sample Input 0**

STDIN         FUNCTION

-----         --------

0101      →    dataStream = "0101"

**Sample Output 0**

2

**Explanation**

The only option is to choose indices 0 and 2 which yields the original string. The maximum length of the non-decreasing substring in the current string is 2.

Sample Case 1

**Sample Input 1**

STDIN         FUNCTION

-----         --------

111000   →    dataStream = "111000"

**Sample Output 1**

5

**Explanation**

One possible way of doing operations can be:

* Choose *i* = 0 and *j* = 4, then the string is "001011".
* Choose *i* = 2 and *j* = 4, then the string is "001110".

Hence, the maximum length of a non-decreasing substring is 5.

**14.Maximum Occurring Character**

Given a string, return the character that appears the maximum number of times in the string. The string will contain only *ASCII* characters, from the ranges ('a'-'z','A'-'Z','0'-'9'), and case matters. If there is a tie in the maximum number of times a character appears in the string, return the character that appears first in the string.

Example

*text = abbbaacc*

Both '*a'* and '*b'* occur *3* times in *text.*  Since '*a'* occurs earlier, *a* is the answer.

Function Description

Complete the function *maximumOccurringCharacter* in the editor below.

*maximumOccurringCharacter* has the following parameter:

*string text:*  the string to be operated upon

Returns

*char :* The most occurring character that appears first in the string.

Constraints

* 10 ≤ |*text*|≤ 104
* All characters are alphanumeric, in the ranges ('a'-'z','A'-'Z','0'-'9')

Input Format For Custom Testing

The first line contains a string, *text*, denoting the text to be analyzed.

Sample Case 0

Sample Input For Custom Testing

STDIN     Function

-----     --------

helloworld →  text = "helloworld"

Sample Output

l

Explanation

The character *'l'* occurs the most, *3* times in the string '*helloworld'*.

Sample Case 1

Sample Input For Custom Testing

STDIN     Function

-----     --------

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz →  text = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz"

Sample Output

a

Explanation

All characters in the string '*abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz'*occur exactly twice. As *'a'* has the lowest index, the character *'a'* is the answer.

**15.1Maximum Score**

An interviewer at HackerRank recently came up with an interesting problem for the candidates.

Given an array *arr*of *n*integers, and an integer *k*, perform *k* operations on the array. Start with a score of 0.

In one operation:

* Choose any element.
* Add its value to the score.
* Replace the element with the integer ceiling of one-third of its value.

For example, if the chosen element is *10,*then *10*is added to the score, and *10*is replaced by *ceil(10 / 3) = 4*.

Find the maximum possible score after *k*operations.

**Example**

Consider *n = 5, arr* = [20, 4, 3, 1, 9], and *k* = 4.

One of the optimal ways to perform the operations is as follows:

1. Choose 20, *score = 20, ceiling(20/3) = 7.*Replace 20 with 7and *arr = [****7****, 4, 3, 1, 9].*
2. Choose 7, *score = 20 + 7 = 27, ceiling(7/3) = 3,* and *arr = [****3****, 4, 3, 1, 9].*
3. Choose 4, *score = 27 + 4 = 31, ceiling(4/3) = 2,* and *arr = [3,****2****, 3, 1, 9].*
4. Choose 9, *score = 31 + 9 = 40, ceiling(9/3) = 3,* and *arr = [3, 4, 3, 1,****3****].*

Bold elements are the elements that are replaced in an operation. Return 40, the score after 4 operations.

**Function Description**

Complete the function *getMaximumScore* in the editor below.

*getMaximumScore* has the following parameters:

*int arr[n]:* the initial array

*int k:* the number of operations to be performed

**Returns**

*long\_int:* the maximum possible score after *k* operations

**Constraints**

* 1 **≤** *n* **≤** 105
* 1 **≤** *arr[i]* **≤** 109
* 1 **≤** *k* **≤** 105

Input Format for Custom Testing

The first line contains an integer *n*, the size of *arr*.

Each of the next *n* lines contains an integer, *arr[i]*.

The next line contains an integer *k*.

Sample Case 0

**Sample Input 0**

STDIN       FUNCTION

-----         --------

4        →   arr[] size, n = 4

4     → arr = [4, 5, 18, 1]

5

18

1

3     → k = 3

**Sample Output 0**

29

**Explanation**

Given *n* = 4, *arr* *= [4, 5, 18, 1], k = 3.*

One of the optimal ways to perform the operations is as follows:

1. Choose 5, *score = 5*and *arr = [4, 2, 18, 1].*
2. Choose 18, *score = 5 + 18 = 23* and *arr = [4, 2, 6, 1].*
3. Choose 6, *score = 23 + 6 = 29* and *arr = [4, 2, 2, 1].*

Sample Case 1

**Sample Input 1**

STDIN      FUNCTION

-----         --------

3        →    arr[] size, n = 3

1        →    arr = [1, 1, 1]

1

1

2        →   number of operations, k = 2

**Sample Output 1**

2

**Explanation**

Given *n* = 3, *arr = [1, 1, 1], k = 2.*

One of the optimal ways to perform the operations is as follows:

1. Choose any element of the array. Then, *score = 1*and *arr = [1, 1, 1].*
2. Choose any element of the array. Then, *score = 1 + 1 = 2* and *arr = [1, 1, 1].*

**16.1Memory Test**

A memory test is being conducted for *n* students standing in a row, numbered from 0 to *n-1* from left to right. The test consists of *m* rounds. In each round, the teacher selects a position *pos[i]* for round *i*. The child at that position will be assigned the number 0. All children to the right of the selected position will be assigned a number that is one greater than the child to their left, and all children to the left of the selected position will be assigned a number that is one greater than the child to their right.

For example, if n = 6 and the selected position is 3, the numbering is [3, 2, 1, 0, 1, 2].

After *m* rounds, the teacher asks each student to shout out the greatest number that was assigned to them during the *m* rounds. Your task is to determine the value each child will shout.

Example

*n* = 5

*m* = 4

*pos =* [3, 0, 1, 4].

| Assigned Numbers per Round | |
| --- | --- |
| pos[i] | Assignments |
| pos[0] = 3 | [3, 2, 1, 0, 1] |
| pos[1] = 0 | [0, 1, 2, 3, 4] |
| pos[2] = 1 | [1, 0, 1, 2, 3] |
| pos[3] = 4 | [4, 3, 2, 1, 0] |

The maximum values each child will shout = [4, 3, 2, 3, 4].

Function Description

Complete the function *maxValues* in the editor with the following parameters:

*int n:* the number of children

*int pos[m]:*  the positions the teacher will call during the *m* rounds

Returns

*int[n]:* the values each child will shout

Constraints

* 1 ≤ *n*≤ 105
* 1 ≤ *m* ≤ 105
* 0 ≤ *pos[i]* ≤ *n -* 1

Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of children.

The next line contains an integer, *m*, that denotes the number of rounds.  
Each line *i* of the *m* subsequent lines contains an integer that describes *pos[i]*.

Sample Case 0

Sample Input For Custom Testing

STDIN Function

----- --------

5 n = 5

3 pos[] size m = 3

2 pos = [2, 0, 3]

0

3

Sample Output

3

2

2

3

4

Explanation

Here, *n = 5, m* = 3, and *pos =* [2, 0, 3].

* After the first round, the numbers assigned = [2, 1, 0, 1, 2].
* After the second round, the numbers assigned = [0, 1, 2, 3, 4].
* After the third round, the number assigned = [3, 2, 1, 0, 1].

Sample Case 1

Sample Input For Custom Testing

STDIN Function

----- --------

4 n = 4

4 pos[] size m = 4

0 pos = [0, 1, 2, 3]

1

2

3

Sample Output

3

2

2

3

Explanation

Here, *n = 4, m* = 4, and *pos =* [0, 1, 2, 3].

* After the first round, the numbers assigned = [0, 1, 2, 3].
* After the second round, the numbers assigned = [1, 0, 1, 2].
* After the third round, the number assigned = [2, 1, 0, 1].
* After the fourth round, the number assigned = [3, 2, 1, 0].

**17.Optimal Change**

Given a binary string *data* of length *n*. An inversion is defined as a pair of indices *(i, j)* where *0 ≤ i < j < n* and *data[i] > data[j]*.

In one operation, you can:

* Flip any bit in the string, changing a *0* to *1* or a *1* to *0*.

You are allowed to perform this operation at most *maxFlips* times. Find the minimum possible number of inversions that can be achieved after making these operations.

**Example**

*maxFlips* = 2

*data* = 0110

The optimal set of operations is:

* Flip the bits at positions 1 and 2 (0-based indexing).

Then, *data* = 0000, which has 0 inversions. Return 0.

**Function Description**

Complete the function *getMinInversions*in the editor with the following parameters:

*string data[n]:* the binary data stream

*int maxFlips:* the maximum number of operations

**Returns**

*int:* the minimum number of inversions

**Constraints**

* 1 ≤ *maxFlips*≤ *n*≤ 5000

Input Format for Custom Testing

The first line contains a string, *data.*

The next line contains an integer *maxFlips*.

Sample Case 0

**Sample Input 0**

STDIN         FUNCTION

-----         --------

1100    →    data = "1100"

1       →    *maxFlips* = 1

**Sample Output 0**

2

**Explanation**

* Flip the bit at position 0 (0-based indexing).

Then, data = 0100, with 2 inversions.

Sample Case 1

**Sample Input 1**

STDIN         FUNCTION

-----         --------

00111    →    data = "00111"

4       →    *maxFlips* = 4

**Sample Output 1**

0

**Explanation**

There is no need to do any operation, as the given string has 0 inversions.

**18.Optimal Selection**

Given an integer array *arr* of length *n*, you can perform the following operation on the array at most once:

* Choose a subset of elements from the array such that no two selected elements are adjacent (i.e., no two selected elements are next to each other in the array).
* Choose a non-negative integer *x*, and increase all the selected elements by *x*.

Determine the minimum non-negative integer *x*, required to make the array *arr* non-decreasing using at most one operation. If it is impossible, return -1.

**Example**

*n* = 4

*arr*= [1, 1, 3, 2]

Some of the possible ways are:

* Select the element with index 3 (0-based indexing) and *increment* = 1 to get the array *arr*= [1, 1, 3, 3].
* Select the elements with indices [1, 3] (0-based indexing) and *increment* = 2 to get the array *arr*= [1, 3, 3, 4].

Hence, the minimum value of increment is 1.

**Function Description**

Complete the function *getMinimumIncrement* in the editor with the following parameter(s):

*int arr[n]:* the array

**Returns**

*int:* the minimum value of *increment* so that the array becomes non-decreasing

**Constraints**

* 1 ≤ *n* ≤ 2 \* 105
* 1 ≤ *arr[i]* ≤ 109

Input Format for Custom Testing

The first line contains an integer *n,* the size of *arr[]*.

Each of the next *n* lines contains an integer *arr[i]*.

Sample Case 0

**Sample Input 0**

STDIN         FUNCTION

-----         --------

5        →    arr[] size n = 5

3        → arr = [3, 1, 3, 2, 4]

1

3

2

4

**Sample Output 0**

2

**Explanation**

One possible way is to select the elements with indices [1, 3] and *increment* = 2 to get the array arr = [3, 3, 3, 4, 4].

Sample Case 1

**Sample Input 1**

STDIN         FUNCTION

-----         --------

6        →    arr[] size n = 6

4        →    arr = [4, 1, 3, 5, 6, 5]

1

3

5

6

5

**Sample Output 1**

-1

**Explanation**

It is impossible to make the array non-decreasing by doing one operation.

**19.Optimal Watchlist**

A streaming platform needs an algorithm to create custom watchlists from a library of *n* movies. Each movie has two key attributes:

* *duration[i]*: The runtime of the *ith* movie in minutes.
* *rating[i]*: The average viewer rating of the *ith* movie.

The watchlist score is calculated as:  
(Total runtime of all movies in the watchlist) × (Minimum rating among all movies in the watchlist).

Determine the maximum possible watchlist score for a watchlist containing at most *limit* number of movies.

Example

*n* = 4

*duration* = [4, 15, 3, 6]

*rating* = [7, 1, 6, 8]

*limit = 3*

| Some possible watchlists containing at most *limit* number of movies. | |
| --- | --- |
| Indices (0-based) of movies in watchlist | Watchlist Score |
| 0, 1, 2 | (4 + 15 + 3) \* min(7, 1, 6) = 22 |
| 0, 3 | (4 + 6) \* min(7, 8) = 70 |
| 0, 2, 3 | (4 + 3 + 6) \* min(7, 6, 8) = 78 |

The maximum possible watchlist score for a watchlist that has at most *limit* movies is 78.

Function Description

Complete the function *getMaxWatchlistScore* in the editor with the following parameters:

*int duration[n]:* The length of the movies in minutes.

*int rating[n]:* The average viewer rating of the movies.

*int limit:* The maximum number of movies allowed in a watchlist.

Returns

*long:* the maximum watchlist score for a watchlist that has at most *limit* movies.

Constraints

* 1 ≤ *limit* ≤ *n* ≤ 2 \* 105
* 1 ≤ *duration[i], rating[i]* ≤ 5 \* 104

Input Format for Custom Testing

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

Each of the next *n* lines contains an integer *duration[i]*.

The next line contains an integer *n*, the size of the array *rating*.

Each of the next *n* lines contains an integer *rating[i]*.

The next line contains an integer *limit*.

Sample Case 0

Sample Input 0

STDIN           Function

-----          --------

5         →     duration[] size n = 5

7         →     duration = [7, 8, 2, 1, 4]

8

2

1

4

5         →     rating[] size n = 5

7         →     rating = [7, 8, 10, 4, 8]

8

10

4

8

3         →     limit = 3

Sample Output 0

133

Explanation

| Some possible watchlists containing at most *limit* number of movies. | |
| --- | --- |
| Indices (0-based) of movies in watchlist | Watchlist Score |
| 1, 2, 4 | (8 + 2 + 4) \* min(8, 10, 8) = 112 |
| 0, 1, 2 | (7 + 8 + 2) \* min(7, 8, 10) = 119 |
| 0, 1, 4 | (7 + 8 + 4) \* min(7, 8, 8) = 133 |

Sample Case 1

Sample Input 1

STDIN           Function

-----           --------

4         →     duration[] size n = 4

1         →     duration = [1, 2, 1, 2]

2

1

2

4         →     rating[] size n = 4

9         →     rating = [9, 7, 6, 10]

7

6

10

2         →     limit = 2

Sample Output 1

28

Explanation

| Some possible watchlists containing at most *limit* number of movies. | |
| --- | --- |
| Indices (0-based) of movies in watchlist | Watchlist Score |
| 0, 3 | (1 + 2) \* min(9, 10) = 27 |
| 3 | (2) \* min(10) = 20 |
| 1, 3 | (2 + 2) \* min(7, 10) = 28 |

**20.Pass the Baton**

There are *n* friends standing in a line, each numbered from *1* through *n* inclusive. The first one, friend *1*, holds a baton. Each second, the baton is passed to the next friend in line. Once it reaches the end of the line, the direction of passing is reversed and passing continues. Determine who will pass and who will receive the baton at a given time.

**Example**

*friends = 4*

*time = 5*

Friends are numbered *1* through *4.*  Friend *1* holds the baton at time *0*. At time *1*, it is passed to friend 2. Over *5* seconds, the baton is passed as 1->2->3->4->3->2. The friend passing the baton at time 5 is friend *3.*  The friend receiving the baton is friend *2.* Return *[3, 2].*

**Function Description**

Complete the function *batonPass* in the editor below. The function must return an integer array.

*batonPass* has the following parameters:

*int friends*: the number of friends

*int time*: the time to report who on the baton

**Returns**

*int[2]:* the friend who has the baton (index 0) and the friend who receives the baton (index 1)

**Constraints**

* *2 ≤  friends ≤ 2 x 105*
* *1 ≤ time ≤ 1012*

Input Format For Custom Testing

The first line contains an integer, *friends*.

The second line contains an integer, *time*, the second to report on the baton.

Sample Case 0

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

5        friends = 5

3        time = 3

**Sample Output**

3

4

**Explanation**

From time 0 to time 3 the baton passes 1->2->3->4. The baton is passed from friend *3* to friend *4* at time *3.*

Sample Case 1

**Sample Input For Custom Testing**

3

6

**Sample Output**

2

3

**Explanation**

The baton is passed as 1->2->3->2->1->2->3. It is passed from friend *2* to friend *3* at time *6.*

**21.Shortest Substring**

Given a string *s* of length *n*. The task is tofind the length of the shortest substring, which upon deletion, makes the resultant string to be consisting of distinct characters only.

A substring is a contiguous sequence of characters within a string. When a substring is deleted, one needs to merge the rest of the character blocks of the string(s). If no substring needs to be deleted, the answer is 0.

**Example**

Consider the given string to be *s = "abcbbk", delete the substring "bb" in the range [3, 4] to get the remaining string "abck" which consists of distinct characters only. This is the minimum possible length of the string. Hence, the answer is 2.*



**Function Description**

Complete the function *findShortestSubstring* in the editor below.

*findShortestSubstring* has the following parameter:

*s:* the given input string

**Returns:**

*int:* an integer representing the length of the shortest substring that should be deleted

**Constraints**

* *1 ≤ n ≤ 105*
* *s consists of lowercase English letters only.*

Input Format For Custom Testing

The first and the only line contains the string *s*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN             FUNCTION

-----             --------

xabbcacpqr    →    s = "xabbcacpqr"

**Sample Output**

3

**Explanation**

Given string *s = "xabbcacpqr".* Considering 0-based indexing, if we delete the substring in the range [3, 5], which is "*bca".* The resulting substring becomes "*xabcpqr"*, in which all characters are distinct. This is the minimum length possible. Thus, the answer is 3.

Sample Case 1

**Sample Input For Custom Testing**

STDIN           FUNCTION

-----           --------

abc        →    s = "abc"

**Sample Output**

0

**Explanation**

The given string *s = "abc",* already contains distinct characters only. So no substring needs to be deleted. Hence, the answer is 0.

**22.2Spam Detection**

Implement a prototype of an email spam detection algorithm.

For simulation, *subjects of n* emails and *k* spam words are given in two arrays of strings, *subjects,* and *spam\_words.* An email is considered spam if it contains at least two spam words in the subject. If a spam word is repeated, it counts as two so the email is considered spam. The spam words are not case-sensitive.

Given *subjects* and *spam\_words,* return an array of *n* strings, "spam" or "not\_spam", one for each subject.

**Example**

**Suppose *subjects* = [“free prize worth millions”, “ten tips for a carefree lifestyle”] and *spam\_words* = [“free”, “money”, “win”, “millions”].**

| **Subject** | **Spam Words** | **Answer** |
| --- | --- | --- |
| **free prize worth millions** | free, millions | spam |
| **ten tips for a carefree lifestyle** | - | not\_spam |

**Hence the answer is ["spam", "not\_spam"].**

**Function Description**

Complete the function *getSpamEmails* in the editor below.

*getSpamEmails* takes the following arguments:

*string subjects[n]:* the subjects of the email

*string spam\_words[k]:* the spam words

**Returns**

*string[n]:* the results of spam detection

**Constraints**

* 1 ≤ n ≤ 103
* 1 ≤ k ≤ 105
* 1 ≤ |subjects[i]| ≤ 105
* 1 ≤ |spam\_words[i]| ≤ 105
* It is guaranteed that the subjects and spam words consist of lowercase and uppercase English letters and spaces only.

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *subjects*.

Each of the next n lines contains a string, *subjects[i]*.

The next line contains an integer, *k*, the number of elements in *spam\_words*.

Each of the next n lines contains a string, *spam\_words[i]*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN FUNCTION

----- --------

2 → subjects[] size n = 2

I paid him paid → subjects = ["I paid him paid", "Summertime Sadness"]

Summertime Sadness

3 → spam\_words[] size k = 3

I → spam\_words = ["I", "Sadness", "paid"]

Sadness

paid

**Sample Output**

spam

not\_spam

**Explanation**

The first subject contains three spam words i.e. 2 "paid" and an "I". The second subject contains only a single spam word "Sadness".

Sample Case 1

**Sample Input For Custom Testing**

STDIN FUNCTION

----- --------

2 → subjects[] size n = 2

Let it go → subjects = ["Let it go", "The right thing to do"]

The right thing to do

5 → spam\_words[] size k = 5

to → spam\_words = ["to", "do", "right", "go", "let"]

do

right

go

let

**Sample Output**

spam

spam

**Explanation**

 The first subject contains two spam words, "go", and "let". Spam words are not case-sensitive. The second one contains three spam words, "to", "do", and "right".

**23.2Stock Prices**

The cost of a stock on each day is given in an array, *arr*. An investor wants to buy the stocks in triplets such that the sum of the cost for three days is divisible by *d*.

The goal is to find the number of distinct triplets *(i, j, k)* such that *i < j < k* and the sum *(arr[i]+arr[j]+arr[k])* is divisible by *d.*

**Example**

Let *arr* = *[3, 3, 4, 7, 8]* and *d = 5.* The triplets whose sum is divisible by *d* are shown.

* Triplet with indices - (0, 1, 2), sum = 3+3+4 = 10
* Triplet with indices - (0, 2, 4), sum = 3+4+8 = 15
* Triplet with indices - (1, 2, 4), sum = 3+4+8 = 15

Hence, the answer is 3.

**Function Description**

Complete the function *getTripletCount* in the editor below. The function must return an integer denoting the total number of distinct triplets.

*getTripletCount* has the following parameters:

*int arr[n]:*  an array of integers

*int d:* the divisor

**Constraints**

* *3 ≤ n ≤ 10 3*
* *1 ≤ arr[i] ≤ 10 9*
* *2 ≤ d ≤ 106*

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr*.

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

The last line contains a single integer, *d*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

4        length of arr[] n = 4

2        arr = [2, 3, 1, 6]

3

1

6

3        d = 3

**Sample Output**

2

**Explanation**

The triplets whose sum is divisible by *d:*

* Triplet with indices - (0, 1, 2), sum = 2 + 3 + 1 = 6
* Triplet with indices - (0, 2, 3), sum = 2 + 1 + 6 = 9

Sample Case 1

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

5        length of arr[] n = 5

5        arr = [5, 3, 5, 5, 8]

3

5

5

8

11       d = 11

**Sample Output**

0

**Explanation**

There is no triplet whose sum is divisible by 11.

**24.Storing Processes**

Developers have ceated a new storage system. There are *n* processes that need to be stored, where the *ith* process is divided into *num\_segments[i]*number of segments. The memory is divided into *m* storage spaces. The segments of processes are stored in the memory spaces in the following way:

* Each segment is stored in exactly one of the memory spaces.
* Each memory space contains at least one process segment.
* A memory space cannot contain segments of different processes.

The *storage efficiency* of the system is defined as the minimum number of segments in a single memory space of all the *m* storage spaces. The developers want to distribute the segments of processes in such a way that the *storage efficiency* of the storage system is maximized. Find the maximum *storage efficiency*.

Example

*n* = 2

*num\_segments* = [7, 10]

*m* = 4,

One of the optimal ways to store the processes is -

* Store 3 segments of process 1 in storage space 1.
* Store 4 segments of process 1 in storage space 2.
* Store 5 segments of process 2 in storage space 3.
* Store 5 segments of process 2 in storage space 4.

The minimum segments in any storage space is 3 in storage space 1. Hence, the maximum possible storage efficiency is 3.

Function Description

Complete the function *getMaximumStorageEfficiency* in the editor below.

*getMaximumStorageEfficiency* has the following parameter(s):

*int num\_segments[n]:*  the number of segments a process is divided into

*long m:* the number of storage spaces

Returns

*int*: the maximum *storage efficiency*

Constraints

* 1 ≤ *n* ≤ 2 \* 105
* 1 ≤ *num\_segments[i]*≤ 109
* *n* ≤ *m* ≤ Σ *num\_segments[i]*

Input Format For Custom Testing

The first line contains an integer, *n*, the number of processes.

Each of the next *n* lines contains an integer *num\_segments[i]*.

The next line contains an integer *m.*

Sample Case 0

Sample Input For Custom Testing

STDIN FUNCTION

-----         --------

3 → num\_segments[] size n = 3

4 → num\_segments = [4, 3, 5]

3

5

3 → m = 3

Sample Output

3

Explanation

As *m* = 3 ( = *n*), the only way to store the processes is -

* Store 4 segments of process 1 in storage space 1.
* Store 3 segments of process 2 in storage space 2.
* Store 5 segments of process 3 in storage space 3.

Hence, the *storage efficiency* is 3.

Sample Case 1

Sample Input For Custom Testing

STDIN FUNCTION

-----         --------

2 → num\_segments[] size n = 2

1 → num\_segments = [1, 2]

2

3 → m = 3

Sample Output

1

Explanation

The only way to store the processes is -

* Store 1 segment of process 1 in storage space 1.
* Store 1 segment of process 2 in storage space 2.
* Store 1 segment of process 2 in storage space 3.

Hence, the *storage efficiency* is 1.

**25.Team Management**

A company needs to divide *n* employees into *k* teams. The skill level of the *ith* employee is given by *empSkill[i]*, and the size of the *jth* team is *teamSize[j]* (for *0 ≤ j < k*).

A team's *strength* is defined as the sum of the minimum and maximum skills of the employees in that team. Determine the maximum possible sum of team strengths, ensuring each employee is assigned to exactly one team.

Example

*n* = 5

*k* = 3

*empSkill* = [1, 10, 5, 9, 9]

*teamSize* = [1, 1, 3].

Some of the team combinations are explained below:

* [1], [10], [5, 9, 9] - Total strength = (1 + 1) + (10 + 10) + (5 + 9) = 36.
* [5], [9], [1, 10, 9] - Total strength = (5 + 5) + (9 + 9) + (1 + 10) = 39.
* [9], [10], [1, 5, 9] - Total strength = (9 + 9) + (10 + 10) + (1 + 9) = 48.

The maximum possible sum of the strengths is 48.

Function Description

Complete the function *getMaximumStrengthSum* in the editor below.

Function Parameters

*int empSkill[n]:* the skills of the employee.

*int teamSize[k]:* the sizes of the teams.

Returns

*int:* the maximum possible sum of strengths of the teams.

Constraints

* 1 ≤ *k* ≤ *n* ≤ 2\*105
* 0 ≤ *empSkill[i]* ≤ 109
* 1 ≤ *teamSize[i]* ≤ *n*
* It is guaranteed that the sum of all *k* team sizes equals *n*.

Input Format for Custom Testing

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

Each of the next *n* lines contains an integer *empSkill[i],* the skills of the employees.

The next line contains an integer *k*, the size of the array *teamSize*.

Each of the next *k* lines contains an integer *teamSize[i],* the size of the *ith* team.

Sample Case 0

Sample Input 0

STDIN           FUNCTION

-----           --------

4         →     empSkill[] size n = 4

1         →     empSkill = [1, 5, 2, 7]

5

2

7

2         →     teamSize[] size k = 2

1         →     teamSize = [1, 3]

3

Sample Output 0

20

Explanation

Some of the possible team combinations:

* [1], [2, 5, 7] - Total strength = (1 + 1) + (2 + 7) = 11.
* [5], [1, 2, 7] - Total strength = (5 + 5) + (1 + 7) = 18.
* [7], [1, 5, 2] - Total strength = (7 + 7) + (1 + 5) = 20.

Sample Case 1

Sample Input 1

STDIN           FUNCTION

-----           --------

6         →     empSkill[] size n = 6

4         →     empSkill = [4, 4, 3, 3, 4, 2]

4

3

3

4

2

3         →     teamSize[] size k = 3

1         →     teamSize = [1, 2, 3]

2

3

Sample Output 1

21

Explanation

Some of the possible team combinations:

* [2], [3, 4], [3, 4, 4] - Total strength = (2 + 2) + (3 + 4) + (3 + 4) = 18.
* [3], [2, 4], [3, 4, 4] - Total strength = (3 + 3) + (2 + 4) + (3 + 4) = 19.
* [4], [4, 3], [3, 4, 2] - Total strength = (4 + 4) + (3 + 4) + (2 + 4) = 21.

**26.Two Strings**

Given two arrays of strings, determine whether corresponding elements contain a common substring.

**Example**

*a = ['ab','cd','ef']*

*b = ['af', 'ee', 'ef']*

Make the following decisions:

i a[i] b[i] Common Result

0 ab af a YES

1 cd ee NO

2 ef ef ef YES

For each test, print the result on a new line, either *YES* if there is a common substring, or *NO*.

**Function Description**

Complete the function *commonSubstring* in the editor below. For each *a[i], b[i]* pair, the function must print *YES* if they share a common substring, or *NO* on a new line.

commonSubstring has the following parameter(s):

*string a[n]:*  an array of strings

*string b[n]:*  an array of strings

**Return**

*void:* output should be printed to stdout (console.log() in javascript) rather than returned

**Constraints**

* All the strings consist of lowercase English letters only, *ascii[a-z]*.
* *|a| = |b|*
* *1 ≤ |a|, |b| ≤ 103*
* *1 ≤ |a[i]|, |b[i]| ≤ 104*

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 *a*.

Each of the next *n* lines contains a string *a[i]* where *0 ≤ i < n*.

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

Each of the next *n* lines contains a string *b[i]* where *0 ≤ i < n*.

Sample Case 0

**Sample Input 1**

STDIN     Function

-----     --------

2     →   a[] size n = 2

hello  →   a = ['hello', 'hi']

hi

2     →   b[] size n = 2

world  →   b = ['world', 'bye']

bye

**Sample Output 1**

YES

NO

**Explanation 1**

i a[i] b[i] Common Output

0 hello world o, l YES

1 hi bye NO

There are two common substrings of *(a[0], b[0])*: '*o'* and '*l'*.

27. **Vowels**

Given a string array that contains *n* elements, each composed of lowercase English letters, and *q* queries, each query of the format *l-r,* for each query, determine how many strings starting from index *l* and ending at index *r* have vowels as the first and last character. Vowels are in *{a,e,i,o,u}.*

**Example**

*strArr = ['aba','bcb','ece','aa','e']*

*queries = ['1-3','2-5','2-2']*

These strings represent two dash delimited integers *l* and *r,* the start and end indices of the interval, inclusive. Using *1*-based indexing in the string array, the interval *1-3* contains two strings that start and end with a vowel: 'aba' and 'ece'. The interval *2-5* also has three. The third interval, from *2-2*, the only element in the interval, 'bcb' does not begin and end with a vowel. The return array for the queries is *[2, 3, 0].*

**Function Description**

Complete the *hasVowels* function in the editor below. It must return an array of integers that represent the result of each query in the order given.

hasVowels has the following parameters.

*strArr string[]:* an array of *n* strings

*query string[]:* an array of *q* strings, each of which describes an interval *l-r* using integers delimited by a dash

**Constraints**

* *1 ≤ n, q ≤ 105*
* *1 ≤ l ≤  r ≤ n*
* *1 ≤ size of strArr[i] ≤ 10*

Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines (where 1 *≤ i ≤ n*) contains a string that describes *strArr[i][i]*.

The next line contains an integer, *q*, denoting the number of elements in *query*.

Each line *j* of the *q* subsequent lines (where *0 ≤ j < q*) contains a string describing *query[j]*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN         Function

-----         --------

5        →   *strArr*[] size n = 5

aab      →   *strArr* = [ "aab", "a", "bcd", "awe", "bbbbbu" ]

a

bcd

awe

bbbbbu

2        →   *query*[] size n = 2

2-3      →   *query =* [ "2-3", "4-5" ]

4-5

**Sample Output**

1

1

**Explanation**

*n = 5*

*strArr = ['aab', 'a', 'bcd', 'awe', 'bbbbbu']*

*q = 2*

*query = ['2-3', '4-5']*

For the first query, *2-3*, only the string at *index 2* has a vowel as the first and last character. For the second query, *4-5*, only the string at *index 4* has vowels as the first and last characters.

Sample Case 1

**Sample Input For Custom Testing**

STDIN         Function

-----         --------

3        →   *strArr*[] size n = 3

yy       →    *strArr* = [ "yy", "u", "oe" ]

u

oe

2        →   *query*[] size n = 2

1-2      →   *query =* [ "1-2", "2-3" ]

2-3

**Sample Output**

1

2

**Explanation**

*n = 3*

*strArr = ['yy', 'u', 'oe']*

*q = 2*

*query = ['1-2', '2-3']*

For the first query, *1-2*, only the string at *index 2* has a vowel as the first and last character. For the second query, *2-3*, both the strings at indices *2* and *3* have vowels as the first and last characters.

**28.Word Count Tool**

Determine the number of valid words in a given string *s*. A valid word contains at least 3 characters with only alphanumeric characters (i.e., the numbers 0-9, letters A-Z in either case), at least one vowel ('a', 'e', 'i', 'o', 'u'), and at least one consonant.

**Example**

Suppose *s =* "This is an example string 234".

| **Word** | **Is Valid** | **Reason** |
| --- | --- | --- |
| This | Yes | At least 3 characters, contains a vowel and a consonant |
| is | No | Less than 3 characters |
| an | No | Less than 3 characters |
| example | Yes | At least 3 characters, contains a vowel and a consonant |
| string | Yes | At least 3 characters, contains a vowel and a consonant |
| 234 | No | Does not contain a vowel or a consonant |

**Function Description**

Complete the function *countValidWords* in the editor below.

*countValidWords* has the following parameter(s):

*string s:*  a string to analyze

**Returns**

*int:* the number of valid words in *s*

**Constraints**

* 1 *≤ |s| ≤*105
* s consists of all available ASCII characters.

Input Format For Custom Testing

The first line contains a string, *s*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN                                 FUNCTION

-----                                 --------

This is Form16 submis$ion date     →  s = "This is Form16 submis$ion date"

**Sample Output**

3

**Explanation**

**Only 'This', 'Form16', and 'date' are valid words. Since 'is' only contains 2 characters and 'submis$ion' has an invalid character, they are not valid.**

Sample Case 1

**Sample Input For Custom Testing**

STDIN                                FUNCTION

-----                                --------

Bob wins the game     →   s = "Bob wins the game"

**Sample Output**

4

**Explanation**

All the words are valid.

**Swap Parity**

Given a number *num* as a string, you can swap two adjacent digits if both digits have the same parity, i.e., both digits are odd or both are even. You can perform this swap operation as many times as needed.

Your task is to find the largest possible number that can be created by applying the swap operation.

Example

Let *num =* "7596801".

* Swap 5 and 9 -> "7956801"
* Swap 7 and 9 -> "9756801"
* Swap 6 and 8 -> "9758601"

The largest value possible is "9758601".

Function Description

Complete the function *getLargestNumber* in the editor below.

*getLargestNumber* has the following parameter:

*string num:* a string of digits

Returns

*string:* the largest number that can be created

Constraints

* 1*≤ length of* *num* *≤*105
* *num*consists of digits 0-9 only.

Input Format For Custom Testing

The first line contains a string, *num*.

Sample Case 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

0082663 → num = "0082663"

Sample Output

8662003

Explanation

Zero is even parity, so swaps can be made until the digits are arranged as shown.

Sample Case 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

5528200 → the given number num

Sample Output

5582200

Explanation

The first two digits, 55, are the only odd parity digits. They cannot be moved. The even parity digits are swapped until they range from high to low.

**Maximum Profit**

An analyst is analyzing *a stock over a period of n* days. The price of the stock on the *ith* day is *price[i],* and the profit obtained is denoted by *profit[i].* The analyst wants to pick a triplet of days *(i, j, k)* such that *(i < j < k)* and *price[i] < price[j] < price[k]* in such a way that the total profit, i.e. *profit[i] + profit[j] + profit[k]* is maximized.

Find the maximum total profit possible. If there is no valid triplet, return -1.

Example

Consider *n* *= 5, price = [1, 5, 3, 4, 6], profit = [2, 3, 4, 5, 6].*

An optimal triplet (considering 1-based indexing) is (3, 4, 5). Here *3 < 4 < 6,* and total profit = 4 + 5 + 6 = 15, the maximum possible. So, the answer is 15.

Function Description

Complete the function *getMaximumProfit* in the editor below.

*getMaximumProfit* has the following parameters:

*int price[n]:* the prices of the stock on each day

*int profit[n]:* the profits obtained from the stock on each day

Returns

*long\_int*: the maximum possible total profit

Constraints

* 1 ≤ *n* ≤ 4000
* 1 ≤ *price[i], profit[i]* ≤ 109

Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *price*.  
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer that describes *price[i].*

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

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

Sample Case 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

5        → price[] size, n = 5

2        → price = [2, 3, 1, 5, 9]

3

1

5

9

5 → profit[] size, n = 5

1 → profit = [1, 2, 6, 1, 5]

2

6

1

5

Sample Output

12

Explanation

An optimal triplet (considering 1-based indexing) is (3, 4, 5). Here *1 < 5 < 9,* and total profit = 6 + 1 + 5 = 12.

Sample Case 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

4        → price[] size, n = 4

4        → price = [4, 3, 2, 1]

3

2

1

4 → profit[] size, n = 4

4 → profit = [4, 3, 2, 1]

3

2

1

Sample Output

-1

Explanation

There is no valid triplet.