Question - 1
Skip A Level

Alex must complete a multi-level game where each level has an entry fee. After completing a level, Alex receives one point. Levels must be played in order, and Alex can skip at most one level.

Given the initial amount in Alex's wallet *k*, the cost of each level in the array *costs[n]*, and there are *n* levels, find the maximum points Alex can collect.

Note: Alex can stop at any point, but can only skip over one level. See Sample 1 for an example.

Example
k = 14
n = 5
costs = [2, 4,1, 8, 6]

Completing the game without skipping any level, entry fees = 2 + 4 + 1 + 8 + 6 = 21 > *k*
Skipping the 4th level, entry fees = 2 + 4 + 1 + 6 = 13 ≤ *k,* points collected = 4, as levels 1, 2, 3, and 5 were completed.

It can be proven that Alex cannot collect more than 4 points. Hence, the answer is 4.

Function Description
Complete the function *maximumPoints* in the editor with the following parameter(s):
int k: the initial number of coins in Alex's wallet
int costs[n]: the costs of each level

Returns
int: the maximum number of points Alex can collect after skipping at most one level

Constraints
- $1 \le k \le 10^9$
- $1 \le n \le 10^5$
- $1 \le costs[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *k*.
The second line contains an integer, *n*, the size of the array *costs*.
Each line *i* of the *n* subsequent lines (where $1 \le i \le n$) contains an integer that describes costs*[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
10      →   k = 10
5       →   n = 5
5       →   costs = [5, 2, 3, 1, 4]
2
3
1
4
```

Sample Output

```
4
```

Explanation

Completing the game without skipping any level, entry fees = 5 + 2 + 3 + 1 + 4 = 15 > *k*

Skipping the 4th level, entry fees = 5 + 2 + 3 + 4 = 14 > *k*

Skipping the 1st level, entry fees = 2 + 3 + 1 + 4 = 10 ≤ *k,* points collected = 4, as levels 2, 3, 4, and 5 were completed.

▼ **Sample Case 1**

Sample Input For Custom Testing

```
STDIN       FUNCTION
----------  -----------
15      →   k = 15
6       →   n = 6
3       →   costs = [3, 2, 6, 4, 6, 1]
2
6
4
6
1
```

Sample Output

```
4
```

Explanation

Completing the last level without skipping any level, entry fees = 3 + 2 + 6 + 4 + 6 + 1 = 22 > *k*

Skipping the 3rd level and stopping after the 5th level, entry fees = 3 + 2 + 4 + 6 = 15 ≤ *k,* points collected = 4, as levels 1, 2, 4, and 5 were completed.

Question - 2

Minimize Array Cost

Given an array of *n* positive integers with 0-based indexing, its cost is calculated as:

$$\sum_{i=1}^{\text{len(arr)}-1} (\text{arr}_i - \text{arr}_{i-1})^2$$

Here, *len(arr)* is the size of the array.

Insert exactly one integer at any position in the array to minimize the cost. Determine the minimum possible cost after this insertion.

Example

a = [1, 3, 5, 2, 10]

The cost of the array before insertion = $(1-3)^2 + (3-5)^2 + (5-2)^2 + (2-10)^2 = 81$.

Two of many scenarios are shown below.

1. Insert 4 between 3 and 5, cost of array = $(1-3)^2 + (3-4)^2 + (4-5)^2 + (5-2)^2 + (2-10)^2 = 79$.
2. Insert 6 between 2 and 10, cost of array = $(1-3)^2 + (3-5)^2 + (5-2)^2 + (2-6)^2 + (6-10)^2 = 49$.

It can be proven that 49 is the minimum cost possible. Return 49.

Function Description

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

int arr[n]: an array of integers

Returns

long_int: the minimum possible cost of the array after inserting one element

Constraints

- $2 \le n \le 10^4$
- $1 \le arr[i] \le 10^5$

▼ Input Format For Custom Testing

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

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

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
4     →   n = 4
4     →   arr = [4, 7, 1, 4]
7
1
4
```

Sample Output

```
36
```

Explanation

The cost of the array before insertion = $(4 - 7)^2 + (7 - 1)^2 + (1 - 4)^2 = 54$.
Insert 5 between 4 and 7, cost = $(4 - 5)^2 + (5 - 7)^2 + (7 - 1)^2 + (1 - 4)^2 = 50$.
Insert 4 between 7 and 1, cost = $(4 - 7)^2 + (7 - 4)^2 + (4 - 1)^2 + (1 - 4)^2 = 36$.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
3     →   n = 3
4     →   arr = [4, 7, 7]
7
7
```

Sample Output

```
5
```

Explanation

The cost of the array before insertion is $(4 - 7)^2 + (7 - 7)^2 = 9$.
Insert 5 between 4 and 7, cost = $(4 - 5)^2 + (5 - 7)^2 + (7 - 7)^2 = 5$.
Insert 6 between 4 and 7, cost = $(4 - 6)^2 + (6 - 7)^2 + (7 - 7)^2 = 5$.

Question - 3
K Smallest Substring

You are given a string *input_str* consisting only of the characters '0' and '1', and an integer *k*. Your task is to find a substring of *input_str* that satisfies the following conditions:

- The substring contains exactly *k* '1's.
- The substring has the smallest possible length.
- Among all substrings of the smallest length, it is lexicographically smallest.

Note: There is always a valid answer.

Example

input_str = "0101101"
k = 3

Some of the possible substrings following the first condition:

- *"01011"*
- *"1101"*
- *"1011"*

The substring that is smallest in length and lexicographically smallest is *"1011"*.

It can be proven that there is no other substring that is smaller than *"1011"* in length and lexicographic order. Hence the answer is *"1011"*.

Function Description

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

string input_str: a string that consists of '0' and '1'

int k: the number of '1's in the answer

Returns

string: the substring that meets the given conditions

Constraints

- $1 \leq k \leq$ *length of* $s \leq 10^3$
- *s[i]* is in the set {'0', '1'}
- The number of '1' characters in string *input_str* is always greater than or equal to *k*.

▼ Input Format For Custom Testing

The first line denotes the given string, *input_str*.
The second line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
10101  → input_str = "10101"
2      → k = 2
```

Sample Output

```
101
```

Explanation

Some of the possible substrings following the first condition:

- *"1010"*
- *"0101"*
- *"101"*

The substring that is smallest in length and lexicographically smallest is *"101"*.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
110011101  →   input_str = "110011101"
4          →   k = 4
```

Sample Output

```
11101
```

Explanation

Some of the possible substrings following the first condition:

- *"110011"*
- *"011101"*
- *"11101"*

The substring that is smallest in length and lexicographically is *"11101"*.

Question - 4

Extraordinary Substrings

Each lowercase English letter has been assigned a digit value as shown in the figure. The numerical value for each letter is its assigned digit.

An *extraordinary* substring is defined as one where the sum of the mapped digit values of its characters is divisible by its length. Given a string *inputStr*, determine the total number of non-empty extraordinary substrings it contains.

Example

inputStr = 'asdf'

All non-empty substrings of *inputStr* 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 with the following parameter(s):

string inputStr: a string of length *n*

Returns

int: the number of non-empty extraordinary substrings

Constraints

- *1 ≤ n ≤ 2000*
- All characters of *inputStr* are lowercase English letters.

▼ Input Format For Custom Testing

The first line contains a string, *inputStr.*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
bdh   →   inputStr = "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  →   inputStr = "abcd"
```

Sample Output

```
6
```

Explanation

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

Question - 5

MEX Problem

Given an array arr containing n non-negative integers and an integer x, you can perform an operation where x can be either added to or subtracted from any element of the array. The MEX of an array is defined as the smallest non-negative integer not present in the array. For instance, for the array [0, 1, 1, 3], the MEX is 2; for the array [1, 2, 4], the MEX is 0.

Determine the highest possible MEX of the array that can be achieved by performing the described operation any number of times.

Example

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

x = 3

If we add *x* to *arr[1]*, we get *arr* = [0, 4, 2, 1, 3], with MEX equal to *5*. This is the maximum possible MEX.

Function Description

Complete the function getM*aximumMex* in the editor with the following parameters:

int arr[n]: an array of *n* non-negative integers

int x: an integer that can be added to or subtracted from any element of the array

Returns

int: the maximum possible MEX of *arr*

Constraints

- $1 \le n \le 10^5$
- $0 \le arr[i] \le 10^9$
- $1 \le x \le 10^5$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the size of the array, *arr*.
Each line *i*, of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *arr[i]*.
The last line contains an integer, *x*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        →    arr[] size n = 3
1        →    arr = [1, 3, 4]
3
4
2        →    x = 2
```

Sample Output

```
2
```

Explanation

By subtracting *x* from *arr[2]* two times, *arr = [1, 3, 0]* that has MEX 2, the maximum possible MEX.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
7        →    arr[] size n = 7
0        →    arr = [0, 1, 2, 2, 0, 0, 10, 3]
1
2
2
0
0
10
3        →    x = 3
```

Sample Output

```
7
```

Explanation

arr can be transformed into *[0, 1, 2, 5, 3, 6, 4, 3]* that has MEX = 7, the maximum possible.

Question - 6
Odd One Out

Analyze a series of strings by checking the difference between adjacent letters of the alphabet. Find the string that has a distinct pattern of differences.

Example

series = ["ACB", "BDC", "CED", "DEF"]:

Calculating the differences between adjacent letters:

- "ACB": ('C'-'A', 'B'-'C') = (+2, -1)
- "BDC": ('D'-'B', 'C'-'D') = (+2, -1)
- "CED": ('E'-'C', 'D'-'E') = (+2, -1)
- "DEF": ('E'-'D', 'F'-'E') = (+1, +1)

The first three strings have the same difference pattern (+2, -1), while "DEF" has a different pattern (+1, +1).
Therefore, "DEF" is the odd one out.

Function Description

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

string series[n]: an array of strings

Returns

string: the odd one out

Constraints

- $3 \le n \le 26$
- 2 ≤ length of *series[i]* ≤ 26
- All strings are uppercase English letters only.
- Within a test case, all strings are of equal length.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *series*.
Subsequent *n lines* of *the* i^{th} element of *series* (where $0 \le i < n$).

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
4       →   series[] size n = 4
ABC     →   series = ["ABC, "BCD", "EFG", "DCB"]
BCD
EFG
DCB
```

Sample Output

```
DCB
```

Explanation

In the series, "ABC", "BCD", "EFG", differences are (+1, +1). For "DCB", differences are (-1, -1).

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
5       →   series[] size n = 5
ADC     →   series = ["ADC", "BED", "CFE", "DGF", "FGH"]
BED
CFE
DGF
FGH
```

Sample Output

```
FGH
```

Explanation
Differences in the first four strings are (+3, -1). The differences in "FGH" are (+1, +1).

Question - 7
Retail Inventory Management

A customer is buying products from an inventory arranged in a row. They use a scooper that picks up three products at a time: the lightest remaining product and its two adjacent products.

In each selection, the customer:

1. Identifies the lightest product (with lowest index if there is a tie)
2. Removes that product and its two adjacent products (or just one if at an edge)
3. Repeats until no products remain

Find the sum of the weights of all the lightest products chosen in each selection.

Example
n = 4 products
weights = [4, 3, 2, 1]

First selection:

- Lightest product is at index 3 with weight 1
- Remove products at indices 2 and 3 (weights 2 and 1)
- Remaining weights: [4, 3]

Second selection:

- Lightest product is at index 1 with weight 3
- Remove products at indices 0 and 1 (weights 4 and 3)
- No products remain

The sum of lightest weights is 1 + 3 = 4.

Function Description
Complete the function *findTotalWeight* in the editor with the following parameters:
products[n]: the array of integers denoting the weights of the products in the inventory

Returns
int: the sum of minimum weighted products at each selection.

Constraints

- $3 \le w \le 2000$
- $3 \le n \le 2000$
- $1 \le products[i] \le 10^5$

▼ Input Format For Custom Testing

The first line contains a single integer, n, the size of *products*.
Each of the next n lines contains an integer, *products[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
7     →     products sizen = 7
6     →     products = [6, 4, 9, 10, 34, 56, 54]
4
9
10
```

```
34
56
54
```

Sample Output

```
68
```

Explanation

First, select the minimum weight, 4. Its adjacent products have weights 6 and 9. Weights 6, 9, and 4 are removed from *products*. The new minimum weight is 10. So, 34 and 10 are removed. At last, the minimum weight is 54. So, 56 and 54 are removed. Hence, the total is 4 + 10 + 54 = 68.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
8     →     products size is 8
132   →     products = [132, 45, 65, 765, 345, 243, 75, 67]
45
65
765
345
243
75
67
```

Sample Output

```
1120
```

Explanation

First, select the minimum weight, 45. Weights 132, 65, and 45 are removed from *products*. The new minimum weight is 67. So, 75 and 67 are removed. The new minimum weight is 243. So, 345 and 243 are removed. At last, the minimum weight is 765. So, 765 is removed. Hence, the total is 45 + 67 + 243 + 765 = 1120.

Question - 8

Customizable Tours with GoWorld

A travel agency organizes customizable tours for traveling enthusiasts. Travelers can choose which places to visit, with all major tourist spots covered in the package.

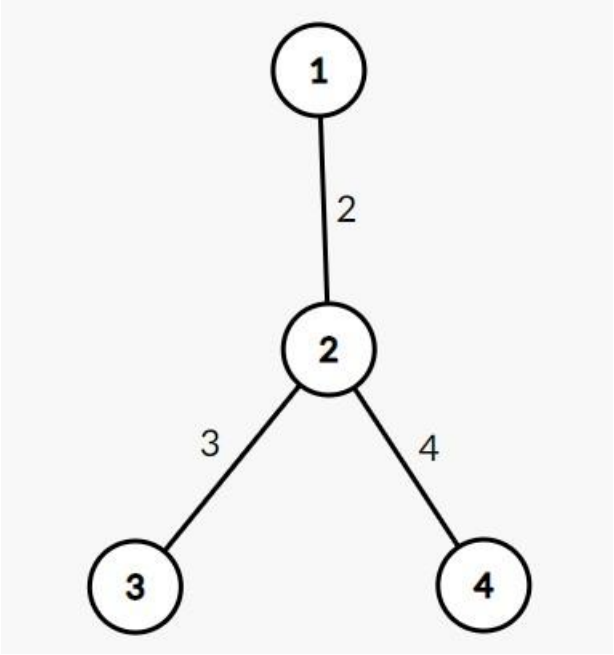
Given a tree-like map of tourist spots, find the minimum cost path that starts from a given starting destination, ends at a specified ending destination, and visits all other nodes at least once.

In this tree:

- Tourist spots are represented as nodes.
- Roads connecting spots are represented as edges with weights (distances).
- An edge may need to be traversed multiple times.
- The cost of traversing an edge with weight w for t times is defined as $w \times \lceil t/2 \rceil$, where $\lceil x \rceil$ is the ceiling value of x.
- The total path cost is the sum of costs for all edges in the path.

Example

tree_nodes = 4
tree_from = [2, 2, 2]
tree_to = [1, 3, 4]
tree_weight = [2, 3, 4]
start = 1
end = 4



The minimum cost path is: 1 → 2 → 3 → 2 → 4

Cost calculation:

- Edge between nodes 1 and 2 (used once): 2 × ⌈1/2⌉ = 2 × 1 = 2
- Edge between nodes 2 and 3 (used twice): 3 × ⌈2/2⌉ = 3 × 1 = 3
- Edge between nodes 2 and 4 (used once): 4 × ⌈1/2⌉ = 4 × 1 = 4

Total cost: 2 + 3 + 4 = 9

Function Description

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

int tree_nodes: number of nodes in the tree
int tree_from[tree_nodes - 1]: one node end of an edge
int tree_to[tree_nodes - 1]: the other node end of an edge
int tree_weight[tree_nodes - 1]: the weights of each edge *i*
int start: denoting the start node of the path
int end: denoting the end node of the path

Returns

int: the value of the minimum cost path, starting at *start*, visiting all nodes at least once, and ending at *end*

Constraints

- $1 \le tree\_nodes \le 10^5$
- $1 \le tree\_from[i], tree\_to[I], start, end \le tree\_nodes$
- $1 \le tree\_weight[i] \le 10^4$

▼ Input Format For Custom Testing

The first line contains two integers, *tree_nodes*, denoting the number of nodes in the tree, and the number of edges in the tree, *tree_nodes - 1*.
The next *tree_nodes - 1* lines each contain three integers denoting *tree_from[i], tree_to[i],* and *tree_weight[i].*
The next line contains an integer, *start*, denoting the start node of the path.
The next line contains an integer, *end*, denoting the end node of the path.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN            FUNCTION
-----            --------
3 2        →     tree_nodes = 3, m = 2
1 2 10     →     there is an edge from 1 to 2 with weight 10
2 3 20     →     there is an edge from 2 to 3 with weight 20
```

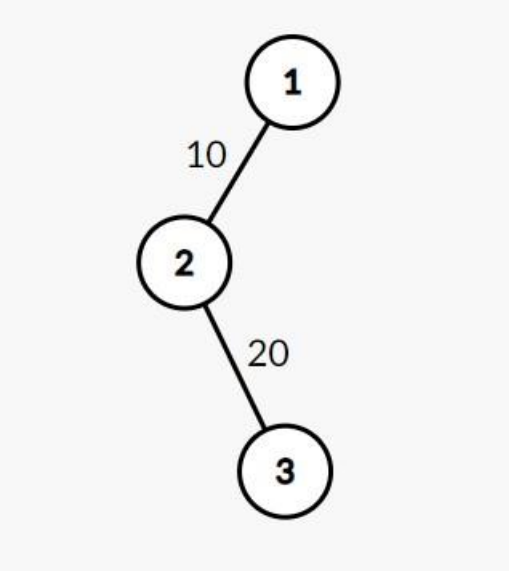
```
1           →     start = 1
3           →     end = 3
```

Sample Output

```
30
```

Explanation

The tree would be:



The minimum cost path is 1 → 2 → 3.

Both the edges 1 → 2 and 2 → 3 are traversed only once. The cost is (10 × ⌈1/2⌉)+ (20×⌈1/2⌉), i.e., 10 + 20 = 30.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN             FUNCTION
-----             --------
2 1         →     tree_nodes = 2, m = 1
1 2 1       →     there is an edge from 1 to 2 with weight 1
1           →     start = 1
1           →     end = 1
```

Sample Output

```
1
```

Explanation

The minimum cost path is 1 → 2 → 1.

Edge 1 → 2 is traversed twice. Therefore, the corresponding cost is 1 × ⌈2/2⌉, i.e., 1.

Question - 9

Array Representations

For each element in an array of *n* integers, determine the minimum value of *k* where the element can be expressed as the sum of *k* integers with these conditions:

- Each of the *k* integers is greater than 0
- Exactly *(k - 1)* integers are even powers of 2 (such as 4, 16, 64)
- Exactly one of the *k* integers is less than 4

If an element cannot be expressed as required, return -1 for that element.

Note: 0 is not considered an even power of 2.

Example

Consider $n = 5$, $arr = [5, 1, 4, 6, 10]$.

- 5 can be expressed as the sum of $1 + 4 = 5$, so $k = 2$.
  - All values are greater than 0.
  - $k - 1 = 1$ value is an even power of 2, i.e. $2^2 = 4$.
  - One value is less than 4.
- 1 can be expressed as 1, so $k = 1$.
  - All values are greater than 0.
  - $k - 1 = 0$ values are an even power of 2.
  - One value is less than 4.
- 4 cannot be expressed in a format that satisfies the conditions. The answer for this element is -1.
  - If $k = 1$, for example, $4 = 4$. Here 4 is an even power of 2, but exactly $k - 1 = 0$ are allowed. There is no value less than 4.
  - If $k = 2$, for example, $2 + 2 = 4$. There is not $k - 1 = 1$ integer that is an even power of 2. More than one integer is less than 4.
  - If $k = 3$, the only numbers that satisfy are $2 + 1 + 1 = 4$. Here $k-1 = 2$ of the values are even powers of 2, but all of the integers are less than 4.
- $2 + 4 = 6$, so $k = 2$.
- $2 + 4 + 4 = 10$, so $k = 3$.

The answer is the array $[2, 1, -1, 2, 3]$.

Function Description

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

int arr[n]: an array of integers

Returns

int[n]: the i^{th} element is the minimum k value for the i^{th} element in *arr* or -1

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le arr[i] \le 2^{30}$

▼ 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 $0 \le i < n$) contains an integer, $arr[i]$.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6         →    arr[] size n = 6
12        →    arr = [12, 3, 7, 8, 15, 19]
3
7
8
15
19
```

Sample Output

```
-1
1
2
-1
4
2
```

Explanation

- 12 cannot be expressed per the conditions, return -1.

- examples: 4 + 8 does not have a value less than 4. 2 + 4 + 6 where $k = 3$ does not contain two even powers of 2.

- 3 can be expressed as 3, hence the value of $k = 1$.
  - There is 1 number less than 4 and $k - 1 = 0$ even powers of 2.
- 3 + 4 = 7, $k = 2$.
- 8 cannot be expressed per the conditions, return *-1*.
- 3 + 4 + 4 + *4* = 15, $k = 4$.
- 3 + 16 = 19, $k = 2$.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4         →    arr[] size n = 4
9         →    arr = [9, 2, 11, 16]
2
11
16
```

Sample Output

```
3
1
3
-1
```

Explanation

Given *n = 4, arr = [9, 2, 11, 16].*

- 1 + 4 + 4 = 9, $k = 3$.
- 2 can be expressed as 2, $k = 1$.
- 3+ 4 + 4 = 11, $k = 3$.
- 16 cannot be expressed per the conditions, return *-1*.

Return the array *[3, 1, 3, -1].*

Question - 10

Unique Decimal Count

Given a binary string consisting only of '0's and '1's, determine how many unique decimal values can be represented by all possible non-empty subsequences of the string.

Notes:

- The binary string may include leading zeros.
- A subsequence is formed by deleting some characters (possibly none) without changing the order of the remaining characters.
- Example: "ace" is a subsequence of "abcde", but "aec" is not.

Example

binary = "010"

- Distinct subsequences of the string are 0, 1, 01, 010, and 10.
- Their corresponding decimal representations are 0, 1, 1, 2, and 2.
- Distinct decimal numbers are 0, 1, and 2.
- The unique decimals count is 3.

Return 3.

Function Description

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

string binary: a binary string

Returns

int: the number of unique decimal representations of binary numbers formed from non-empty subsequences of the string

Constraints

- 1 ≤ length of *binary* ≤ *20*
- The string *binary* consists of characters '0' and '1' only

▼ Input Format For Custom Testing

The first line contains a binary string, *binary*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
011       →     binary = "011"
```

Sample Output

```
3
```

Explanation

- Distinct subsequences of the string are 0, 1, 01, and 11.
- Their corresponding decimal representations are 0, 1, 1, and 3.
- Distinct decimal numbers are 0, 1, and 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
11        →     binary = "11"
```

Sample Output

```
2
```

Explanation

- Distinct subsequences of the string are 1, and 11.
- Their corresponding decimal representations are 1, and 3.
- Distinct decimal numbers are 1, and 3.

Question - 11

Bracket Sequence

Given a bracket sequence of length *n* (where *n* is odd), determine how many positions exist where inserting a single bracket '(' or ')' would create a valid bracket sequence.

A "correct bracket sequence" can be transformed into a correct arithmetic expression by inserting characters "1" and "+" between the original brackets. Examples of correct sequences include "()()"->(1)+(1)" and "(())"->((1+1)+1)", while ")(" and "(" are invalid. The empty string is correct by definition.

Example

Let the string s = "())(". Possible brackets sequences after insertion are:

- Insert the bracket '(' before index 1, that is, (()(), which is a correct bracket sequence
- Insert the bracket '(' before index 2, that is, (()(), which is a correct bracket sequence
- Insert the bracket '(' before index 3, that is, ()(), which is a correct bracket sequence
- Insert the bracket '(' before index 4, that is, ()(), which is a correct bracket sequence
- Insert the bracket '(' before index 5, that is, ()()(), which is a correct bracket sequence
- Insert the bracket '(' at the end, that is, ())((, which is not a correct bracket sequence

Hence, the number of ways to make the above string a correct bracket sequence is 5.

Function Description

Complete the function countBracketSequence in the editor with the following parameter:

string s: the given bracket sequence

Returns

int: the number of ways to make the string a *correct bracket sequence*

Constraints

- $1 \leq n \leq 10^6$
- s contains only '(' and ')'
- It is guaranteed that the length of the string s is odd

▼ Input Format For Custom Testing

The only line of input contains a string s denoting the bracket sequence.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
())    -> s = "())"
```

Sample Output

```
3
```

Explanation

The following are the possible insertion operations:

- Insert the bracket '(' before index 1, that is, (()), which is a correct bracket sequence
- Insert the bracket '(' before index 2, that is, (()), which is a correct bracket sequence
- Insert the bracket '(' before index 3, that is, ()(), which is a correct bracket sequence
- Insert the bracket '(' at the end, that is, ())(, which is not a correct bracket sequence

Hence, the number of ways to make the string a correct bracket sequence is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
(((    -> s = "((("
```

Sample Output

```
0
```

Explanation

It is not possible to make it a correct bracket sequence by adding an extra '(' or ')'.

Question - 12
Astronomy Board Game

In an astronomy-themed board game, there are *n* planets arranged in a row within an imaginary universe where typical gravitational laws do not apply. The system is stable only when the sum of the masses of planets at even positions equals the sum of the masses of planets at odd positions.

Currently, the system is unstable, but a player can achieve stability by destroying one planet. Return the 1-based index of the planet that should be destroyed to stabilize the system. If no such planet exists, return -1. If multiple planets can be destroyed to achieve stability, choose the one with the smallest index.

Example

n=5
planets = [2,4,6,5,4]

Destroying the planet at the 1-based index 4 of mass 5 results in *planets* = [2, 4, 6, 4]. The sum of odd-index planets is 2 + 6 = 8, which equals the sum of even-index planets, 4 + 4 = 8. Hence, return 4.

Function Description

Complete the function *getPlanetToDestroy* in the editor with the following parameter(s):
int planets[n]: the planets' masses

Returns
int: the 1-based index of the planet to destroy

Constraints
- $2 \leq n \leq 2 \times 10^5$
- $1 \leq planets[i] \leq 10^9$

▼ Input Format For Custom Testing

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

Each line *i* of the next *n* lines (where $1 \leq i \leq n$) contains an integer *planets[i].*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
4      -> planets[] size n = 4
2      -> planets[] = [2, 5, 3, 1]
5
3
1
```

Sample Output

```
2
```

Explanation
If we destroy the second planet of mass 5, then *planets* = [2, 3, 1].

odd sum = 2 + 1 = 3
even sum = 3

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     FUNCTION
-------   -----------
6      -> planets[] size n = 6
2      -> planets[] = [2, 5, 6, 7, 8, 4]
5
6
7
8
4
```

Sample Output

```
3
```

Explanation

If we destroy the third planet of mass 6, then *planets* = [2, 5, 7, 8, 4].

odd sum = 2 + 7 + 4 =13
even sum = 5 + 8

Question - 13

Series Solver

Given an array of strings, all but one will exhibit a consistent pattern in the differences between adjacent character values. Identify the string that deviates from this pattern, the 'odd one out'.

Example

Given *series = ['ACB', 'BDC', 'CED', 'DEF']* , the distances for ACB, BDC, and CED are (2, -1). For DEF, the distances are (1, 1), so it is the odd one out.

Function Description

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

string series[n]: an array of strings

Returns

string: the odd one out

Constraints

- $3 \le n \le 26$
- $2 \le length\ of\ element \le 26$
- All strings contain only uppercase English letters ascii[A-Z]
- All strings in a single test case will be the same length.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *series*.
Each of the subsequent *n* lines is the i^{th} element of series (where $3 \le i < n$).

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
----------      ----------------------
4        ->     series[] size n = 4
ABCD     ->     series[] = [ "ABCD" , "BCDE" , "EFGH" , "DCBE" ]
BCDE
```

```
EFGH
DCBE
```

Sample Output

```
DCBE
```

Explanation

The strings ABCD, BCDE, and EFGH have distances of (1, 1, 1). DCBE has differences of (1, -1, 3).

▼ **Sample Case 1**

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----------    --------------------
5        ->    series[] size n = 5
ADC      ->    series[] = [ "ADC" , "BED" , "CFE" , "DGF" , "FGH" ]
BED
CFE
DGF
FGH
```

Sample Output

```
FGH
```

Explanation

ADC, BED, CFE, DGF -> (3, -1)

FGH -> (1, 1)

Question - 14

Alternating Prime Sequence

An alternating prime sequence is a sequence where adjacent elements alternate between prime and non-prime. For example, [1, 2, 4, 3, 6, 5] is a valid alternating prime sequence because the elements alternate between non-prime and prime. However, [1, 2, 3, 4, 5, 6] is not valid as 2 and 3 are both primes and they are adjacent.

A student needs to construct an alternating prime sequence from an array of integers for a coding challenge. If an integer cannot be used, its value is added to a penalty accumulator.

Your task is to find the minimum possible penalty that the student can have after completing the task.

Example

n = 6 elements in *arr*

arr = [3, 7, 1, 4, 6, 6]

The student can construct a sequence [4, 3, 6, 7, 6], which has a penalty of 1. Note that there are other possible sequences as well, but 1 is the minimum penalty.

Function Description

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

int arr[n]: an array of integers

Returns

int: the minimum penalty value

Constraints

- $1 \le n \le 10^4$
- $1 \le arr[i] \le 10^4$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *arr*.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
2          -> arr[] size n = 2
50         -> arr[] = [ 50, 50 ]
50
```

Sample Output

```
50
```

Explanation
Since both are non-prime, the only valid sequence is [50]. Hence, the penalty is 50.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
6          -> arr[] size n = 6
1          -> arr[] = [ 1, 2, 3, 4, 5, 6 ]
2
3
4
5
6
```

Sample Output

```
0
```

Explanation
The sequence [1, 2, 4, 3, 6, 5] has 0 penalty.

Question - 15
Reduce the Array

Given an integer array, reduce it to a single element by repeatedly performing the following operation:

1. Pick two indices *i* and *j* (where $i \neq j$)
2. Append the value *a[i]* + *a[j]* to the array
3. Delete *a[i]* and *a[j]* from the array

The cost of each operation is *a[i]* + *a[j]*. Find the minimum possible total cost to reduce the array to a single element.

Example
array = [25, 10, 20]

Return: 85

Explanation:

- Pick 10 and 20, cost = 30, *array* becomes [25, 30]
- Pick 25 and 30, cost = 55, *array* becomes [55]
- Total cost = 30 + 55 = 85

Function Description

Complete the function minimizeCost in the editorwith the following parameters:

int arr[n]: an array of integers

Returns

int: the minimum cost of reducing the array

Constraints

- $2 \leq n \leq 10^5$
- $1 \leq arr[i] \leq 100$

▼ 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 $0 \leq i < n$) contains a single value, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3        ->   arr[] size n = 3
30       ->   arr[] = [ 30, 10, 20 ]
10
20
```

Sample Output

```
90
```

Explanation

- Pick 10 and 20, cost = 10+20 = 30, array' = [30,30].
- Pick 30 and 30, cost = 30+30 = 60, array'' = [60].

The total cost is 30+60 = 90.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
2        ->   arr[] size n = 2
100      ->   arr[] = [ 100, 1 ]
1
```

Sample Output

```
101
```

Explanation

Only one operation is performed, with 100 and 1, and the cost is 100+1 = 101.

Question - 16

Closest Numbers

Given an array of distinct integers, determine the minimum absolute difference between any two elements. Print all pairs of elements with that difference, ensuring the smaller number appears first in each pair, and the pairs are sorted in ascending order.

Example

numbers = [6,2,4,10]

The minimum absolute difference is *2* and the pairs with that difference are *(2,4)* and *(4,6)*.

```
2 4
4 6
```

Function Description

Complete the function *closestNumbers* in the editor.

closestNumbers has the following parameter(s):

int numbers[n]: an array of integers

Returns

NONE

Prints

distinct element pairs that share the minimum absolute difference, displayed in ascending order with each pair separated by one space on a single line

Constraints

- $2 \leq n \leq 10^5$
- $-10^9 \leq numbers[i] \leq 10^9$

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

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-------     -----------
4           →   numbers[] size n = 4
4           →   numbers = [4, 2, 1, 3]
2
1
3
```

Sample Output 0

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

Explanation 0

The minimum absolute difference between any two elements in the array is *1*, and there are three such pairs with this difference: *(1, 2)*, *(2, 3)*, and *(3, 4)*.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-------     -----------
4       →   numbers[] size n = 4
```

```
4       →   numbers = [4, -2, -1, 3]
-2
-1
3
```

Sample Output 1

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

Explanation 1

The minimum absolute difference between any two elements in the array is *1*, and there are two such pairs: *(-2, -1)* and *(3, 4)*.

Question - 17

Throw The Ball

Track a ball being passed between friends after a specific time.

Task: You are given:

- An array *receiver* of length *n* where *receiver[i]* indicates which friend the i^{th} friend throws the ball to.
- An integer, *k,* representing the number of seconds.

Determine which friend (numbered 1 to *n*) has the ball after exactly *k* seconds have passed, assuming:

1. Friends are numbered starting with 1.
2. Friend 1 starts with the ball.
3. Each friend throws the ball to another friend every second.

Example

receiver = [2 , 4 , 1 , 5 , 3]
k = 6

Second	Has ball	Receiver
1	1	2
2	2	4
3	4	5
4	5	3
5	3	1
6	1	2

After 6 seconds, the ball will be with friend 2.

Function Description

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

int receiver[n]: the i^{th} friend will throw the ball to the friend indicated in *receiver[i]*
int k: the time in seconds that the game lasts

Returns

int: the friend holding the ball at *time = k*

Constraints

- $2 \le n \le 2 \times 10^5$
- $1 \le receiver[i] \le n$ $(receiver[i] \ne i)$
- $1 \le k \le 10^{12}$

▼ Input Format For Custom Testing

The first line contains an integer, *n,* the size of *receiver.*
Each of the next *n* lines contains an integer, *receiver[i].*
The last line contains an integer, *k.*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
4         receiver[] size n = 4
3         receiver = [3, 1, 4, 2]
1
4
2
5         k = 5
```

Sample Output

```
3
```

Explanation

The ball is thrown in the following sequence, beginning with 1 at time = 0: 1-3-4-2-1-3.

▼ Sample Case 1

Sample Input For Custom Testing

```
6
6
5
2
5
3
2
7
```

Sample Output

```
3
```

Explanation

The ball is thrown in the following sequence: 1-6-2-5-3-2-5-3.

Question - 18
The Missing Prisoner

Prisoners are positioned on a Cartesian coordinate system inside a jail, each at a unique integer coordinate point *[x, y]*. Each prisoner must be positioned such that they form a rectangle or square with three other prisoners.

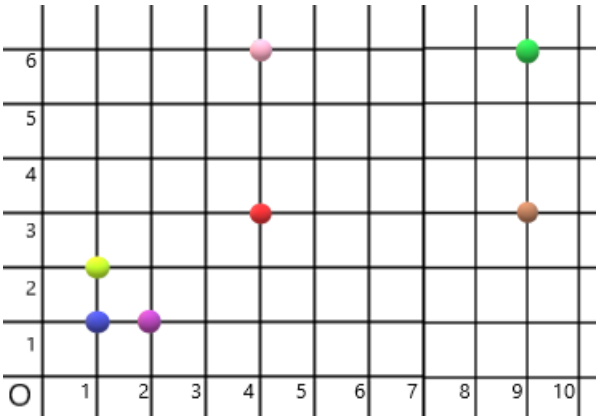
Specifically, for any prisoner at coordinates *[x, y]*, there must be:

- At least one prisoner at coordinates *[x, ay]* where $ay \neq y$
- At least one prisoner at coordinates *[bx, y]* where $bx \neq x$
- At least one prisoner at coordinates *[bx, ay]*

One prisoner has escaped. Determine the coordinates of the escaped prisoner.

Example

locations = [[1, 1], [1, 2], [2, 1], [4, 4], [4, 6], [9, 4], [9, 6]]



Per the rules, each group of *4* prisoners must occupy positions *[x, y], [bx, y], [bx, ay], [x, ay].* The prisoner is missing at *[2, 2].*

Function Description

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

int locations[n][2]: coordinates of the prisoners who are present

Returns

int[2]: the *[x, y]* coordinates of the missing prisoner

Constraints

- $3 \leq n \leq 8 \times 10^5$
- $-2 \times 10^9 \leq x[i], y[i] \leq 2 \times 10^9$

▼ Input Format For Custom Testing

The first line of input contains an integer, *n* (the number of prisoners whose coordinates are known, and the size of *locations[]*).

The next line contains an integer, *2*, the size of *locations[n][]*, one position for *x[i]* and for *y[i]*.

Each of the next *n* lines contains *2* space-separated integers, *x[i]* and *y[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

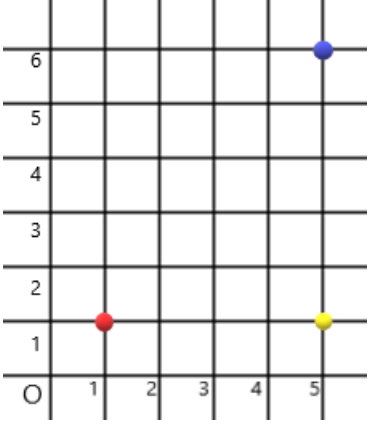
```
STDIN       Function
-----       --------
3           locations[] size n = 3
2           locations[n][] size (always 2)
1 1         locations =[[1, 1], [5, 1], [5, 6]]
5 1
5 6
```

Sample Output

```
1
6
```

Explanation

The *3* prisoners are at the locations shown below.



The escaped prisoner should be at coordinates [1, 6].

▼ Sample Case 1

Sample Input For Custom Testing

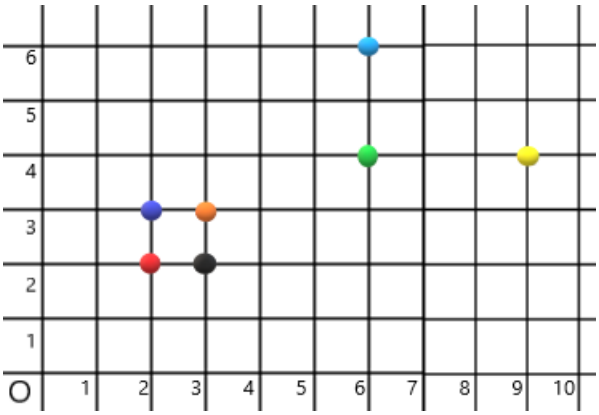
```
2
7
2
6 4
2 2
3 2
6 6
9 4
2 3
3 3
```

Sample Output

```
9
6
```

Explanation

There should be *2* groups of *4* prisoners. The locations of the *7* prisoners are illustrated below.



The escaped prisoner should be at *[9, 6]*.

Question - 19

Linked List Creation

Create a new linked list by repeatedly selecting nodes from odd positions in an original singly linked list. Each node has an integer value (*data*) and a pointer to the next node (*next*).

Follow these steps:

1. Select all nodes at odd positions from the original list.
2. Append these nodes to a new linked list, preserving their original order.
3. Remove these nodes from the original list.
4. Repeat steps 1-3 until the original list is empty.

Return a reference to the head of the new linked list.

Note: Do not use extra memory except for creating new nodes.

Example

The initial linked list is: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

Move nodes in odd positions to the new linked list:

- Original list: $2 \rightarrow 4$
- New list: $1 \rightarrow 3 \rightarrow 5$

Repeat the process:

- Original list: (empty)
- New list: $1 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 4$

Return a reference to the head of this list.

Function Description

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

head: reference to the head of a linked list of *n* integers

Returns

a reference to the head of the final linked list

Constraints

- $1 \le n \le 10^5$
- $1 \le data \le 10^9$

▼ Input Format For Custom Testing

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

Each *i* of the next *n* lines contains an integer, the i^{th} element of the list.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
5             list size n = 5
3             list = 3 → 5 → 3 → 7 → 8
5
3
7
8
```

Sample Output

```
3
3
8
5
7
```

Explanation

- The new list is $3 \rightarrow 3 \rightarrow 8$, the old list is $5 \rightarrow 7$.
- $3 \rightarrow 3 \rightarrow 8 \rightarrow 5$ and 7
- $3 \rightarrow 3 \rightarrow 8 \rightarrow 5 \rightarrow 7$ and empty

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
4             list size n = 4
5             list = 5 → 3 → 2 → 7
3
2
7
```

Sample Output

```
5
2
3
7
```

Explanation

- The new list is 5 → 2 and the old list is 3 → 7.
- 5 → 2 → 3 and 7
- 5 → 2 → 3 → 7 and empty

Question - 20
Women who Code

Alaina is preparing a question for her friends in her Women Who Code college chapter for a hackathon. In this task, Alaina provides an integer array, and her friends are required to find the top *k* integers with maximum bit distances.

The bit distance of an integer is defined as the maximum distance between any two consecutive set bits in its binary representation. If an integer has only one set bit, its bit distance is -1.

Your task is to sort the given integers in decreasing order of their bit distances. If multiple integers have the same bit distance, sort them in decreasing order by their value. Then return the first *k* integers from this sorted array.

Example
numbers = [12, 4, 5, 10, 8]
k = 3

The binary representations are:

- 12 = 1100 (bit distance = 0)
- 4 = 100 (bit distance = -1)
- 5 = 101 (bit distance = 1)
- 10 = 1010 (bit distance = 1)
- 8 = 1000 (bit distance = -1)

Sorting by bit distance (descending) and then by value (descending):

- 10 (bit distance = 1, value = 10)
- 5 (bit distance = 1, value = 5)
- 12 (bit distance = 0, value = 12)
- 8 (bit distance = -1, value = 8)
- 4 (bit distance = -1, value = 4)

The top *k* = 3 integers are [10, 5, 12].

Function Description
Complete the function *getTopKBitDistances* in the editor with the following parameter(s):

int numbers[n]: an array of positive integers
int k: the number of integers to select

Returns

int[k]: the topmost *k* integers in decreasing order of their bit distances and integer values

Constraints

- $1 \le n \le 10^5$
- $1 \le numbers\ [i] \le 10^8$
- $1 \le k \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *numbers*.
Each of the *n* subsequent lines (where $0 \le i < n$) contains a positive integer, *numbers[i]*.
The next line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
3       →   numbers size is 3
3       →   numbers = [3, 5, 8]
5
8
1       →   k = 1
```

Sample Output

```
5
```

Explanation

The integers in binary form are [11, 101, 1000].
The corresponding *d* values are [0, 1, -1].
Because *k* = 1, only the integer with the highest bit distance is reported, which is 5.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
4       →   numbers size is 4
10      →   numbers = [10, 13, 5, 18]
13
5
18
3       →   k = 3
```

Sample Output

```
18
13
10
```

Explanation

The integers in binary form are [1010, 1101, 101, 10010]. The corresponding bit distances are [1, 1, 1, 2].
18 is reported first because it has the highest bit distance.
10, 13, and 5 all have the same bit distance. Next, 13 and 10 are reported in that order because those are the two largest integers in decreasing order.

Question - 21
Madam C.J. Walker's Business Plan

Madam C.J. Walker, known as the first African-American businesswoman in America, had a business model involving the sale of cosmetics and perfumes for women, which can be likened to a modified 0-1 knapsack problem.

There are n different products available for purchase and resale. The cost of the i^{th} item is given by *cost[i]*, and it can be resold for a profit of 2^i.

Given the total available to x to invest, determine the maximum profit that can be generated with the given amount of money. Since the answer can be quite large, return the result modulo (10^9+7).

Example

Suppose there are $n = 5$ items with costs *cost* = [10, 20, 14, 40, 50], and Walker's initial amount of money is $x = 70$.

Some combinations of items (0-based indexing) Walker can buy are:

- Items 0, 1, and 2 for a cost of 44 and obtain a profit of $2^0 + 2^1 + 2^2 = 7$.
- Items 0 and 4 for a cost of 60 and obtain a profit of $2^0 + 2^4 = 17$.
- Items 1 and 4 for a cost of 70 and obtain a profit of $2^1 + 2^4 = 18$.
- Items 2 and 4 for a cost of 64 and obtain a profit of $2^2 + 2^4 = 20$.

Out of all the possible combinations, the maximum profit is 20 when purchasing items 2 and 4 for a cost of 64.

Function Description

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

int cost[n]: the costs of the items

int x: the amount of money available

Returns

int: the maximum profit possible modulo (10^9+7).

Constraints

- $1 \le n \le 10^5$
- $1 \le cost[i] \le 10^5$
- $0 \le x \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, n, denoting the number of elements in *cost*.
Each line i of the n subsequent lines (where $0 \le i < n$) contains an integer describing *cost[i]*.
The next line contains an integer, x.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
3       →   cost size is 3
3       →   cost = [3, 4, 1]
4
1
8       →   x = 8
```

Sample Output

```
7
```

Explanation

Walker can buy all the items since their total cost is less than or equal to her available money.
The total profit is $2^0 + 2^1 + 2^2 = 7$ modulo $(10^9+7) = 7$.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
----------  -----------
5       →   cost size is 5
19      →   cost = [19, 78, 27, 18, 20]
78
27
18
20
25      →   x = 25
```

Sample Output

```
16
```

Explanation

The optimal solution is to buy the last item for a profit of $2^4 = 16$, and 16 modulo $(10^9+7) = 16$.

Question - 22

K-th Occurrence Queries

Find the positions of specific occurrences of a value in an array.

You are given:

- An array of integers, *arr*.
- An integer *X*.
- An array of integers, *queryValues*.

For each element *queryValues[i]*, find the 1-based index of the *queryValues[i]*th occurrence of *X* in *arr*. If *X* does not appear *queryValues[i]* times, return -1 for that query.

Example

arr = [1, 2, 20, 8, 8, 1, 2, 5, 8, 0]

X = 8

queryValues = [100, 4, 2]

There is no 100th or 4th instance of *X* = 8 in *arr.* The 2nd occurrence is at index 5. Return [-1, -1, 5].

Function Description

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

int X: the integer to search for

int arr[n]: the elements to search

int queryValues[q]: the occurrence of X to find the index of

Returns

int[q]: the index in *arr* or -1 for each query

Constraints

- *1 <= n, q <= 2 * 10^5*
- *0 <= arr, X, queryValues <= 10^9*

▼ Input Format For Custom Testing

The first line contains an integer, the value of *X*.
The next line contains a single integer *n*, the size of *arr*.
The next *n* lines each contain an integer, *arr[i]*.
The next line contains a single integer *q*, the size of *queryValues*.
The next *q* lines each contain an integer, *queryValues[j]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
8         ->  X = 8
10        ->  arr[] size n = 10
1         ->  arr[] = [1, 2, 20, 8, 8, 1, 2, 5, 8, 0]
2
20
8
8
1
2
5
8
0
5         ->  queryValues[] size q = 5
100       ->  queryValues[] = [100, 2, 1, 3, 4]
2
1
3
4
```

Sample Output

```
-1
5
4
9
-1
```

Explanation

There is no 100^{th} or 4^{th} occurrence of 8, hence, for those queries, the answer is -1. For the rest, it is the index of the first, second, and third occurrences of *X* = 8.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
9         ->  X = 9
4         ->  arr[] size n = 4
9         ->  arr[] = [9, 8, 9, 9]
8
9
9
4         ->  queryValues[] size q = 4
7         ->  queryValues[] = [7, 3, 7, 6]
3
7
6
```

Sample Output

```
-1
4
-1
-1
```

Sample Explanation

There is no 6^{th} or 7^{th} occurrence of 9. The 3^{rd} occurrence is at index 4.

Question - 23

Statistic Indicators

Given an array, find two statistical indicators and report the absolute difference between them:
Indicator 1: The number of instances where a number *k* appears for exactly *k* consecutive times in the array.
Indicator 2: The number of instances where a number *k* appears for exactly *k* consecutive times in the array, starting from index *k* (using 1-based indexing).

Examples

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

- For indicator 1
  - 1(1), 2(4), 3(3), 1(2), 2(2) where the values in parentheses represent consecutive occurrences.
  - Indicator 1 equals 3, corresponding to 1(1), 3(3), and 2(2).
- For indicator 2
  - the only value that qualifies is 1(1) at index 1
  - Indicator 2 equals 1.
- Their absolute difference is 2.

If *arr* = [2, 2, 2, 4, 4, 4, 4, 4, 4]:

- For indicator 1
  - 2(3), 4(6), so no counts match.
  - Indicator 1 equals 0.
- For indicator 2
  - At index 2, we have exactly 2 consecutive 2s.
  - At index 4, we have 6 consecutive 4s.
  - Indicator 2 equals 1, corresponding to index 2.
- Their absolute difference is 1.

Your task is to find the absolute difference between these two indicators.

Function Description

Complete the function *difference_calculator* in the editor with the following parameter(s):
int arr[n]: the array of integers

Returns

int: the absolute difference between the indicators

Constraints

- $1 \le n \le 100$
- $1 \le arr[i] \le 15$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the size of *arr*.
Each of the next *n* lines contains an integer, *arr[i]*, where $1 \le i \le n$.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
14         -> arr[] size n = 14
3          -> arr[] = [ 3, 3, 2, 2, 5, 5, 5, 5, 5,  3, 3, 3, 2, 2 ]
3
2
2
5
5
5
5
5
```

```
3
3
3
2
2
```

Sample Output

```
3
```

Explanation

Indicator 1 is 4.

- 3 3 2 2 5 5 5 5 5 3 3 3 2 2
- 2 appears exactly two consecutive times.
- 5 appears exactly five consecutive times.
- Then, 3 appears exactly three consecutive times.
- Then, 2 appears exactly two consecutive times.

Indicator 2 is 1.

- 3 3 2 2 5 5 5 5 5 3 3 3 2 2
- 5 appears exactly five times, starting at index 5.
- The difference is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
10        -> arr[] size n = 10
1         -> arr[] = [ 1, 2, 2, 4, 4, 4, 4, 2, 2, 2 ]
2
2
4
4
4
4
2
2
2
```

Sample Output

```
0
```

Explanation

Indicator 1 will be 3.

- 1 2 2 4 4 4 4 2 2 2
- 1 appears exactly once.
- 2 appears exactly two consecutive times.
- 4 appears exactly four consecutive times.

Indicator 2 will be 3.

1 2 2 4 4 4 4 2 2 2

- 1 appears exactly once, starting from index 1.
- 2 appears exactly two consecutive times, starting from index 2.
- 4 appears exactly four consecutive times, starting from index 4.
- The difference is 0.

Question - 24

Highly Profitable Months

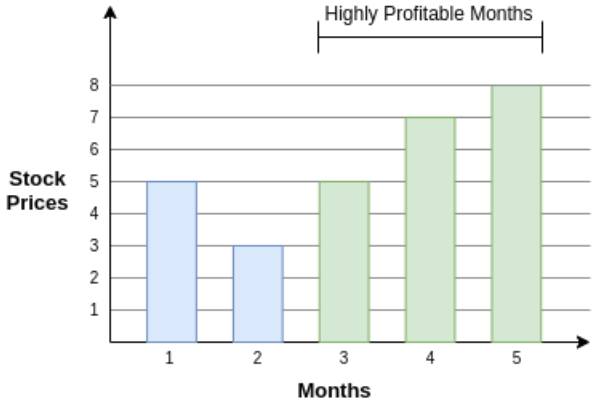
The stock performance of a company is being examined to assess its net profit over time.

For a given analysis parameter *k*, an interval of *k* consecutive months is considered *highly profitable* if the stock prices increase strictly throughout those months. Given the stock prices for *n* months and the analysis parameter *k*, determine the number of such highly profitable intervals.

Example

stockPrices = [5, 3, 5, 7, 8]
k = 3

These are the intervals of *k* months in which the stock prices are strictly increasing:



Hence the answer is 2.

Note: If the interval length is 1, each subarray of length 1 is highly profitable.

Function Description

Complete the function *countHighlyProfitableIntervals* in the editor below.

countHighlyProfitableIntervals has the following parameters:

int stockPrices[n]: the stock prices for *n* months

int k: the analysis parameter

Returns

int: the number of highly profitable intervals

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le stockPrices[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in the array *stockPrices*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *stockPrices[i]*.
The last line contains an integer, k, denoting the analysis parameter.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
6        →     stockPrices[] size, n = 6
1        →     stockPrices = [1, 2, 3, 3, 4, 5]
2
```

```
3
3
4
5
3        →    k = 3
```

Sample Output

```
2
```

Explanation

- Months 1 to 3: [1, 2, 3]
- Months 4 to 6: [3, 4, 5]

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     stockPrices[] size, n = 4
1        →     stockPrices = [1, 2, 3, 4]
2
3
4
4        →     k = 4
```

Sample Output

```
1
```

Explanation

Since *k* is equal to *n*, we can only make one group consisting of all the months. Since this is strictly increasing, the answer is 1.

Question - 25

Balance Parentheses

Given a sequence of parentheses, determine the minimum number of swaps needed to make the sequence balanced. Each opening bracket must have a corresponding closing bracket. Swapping adjacent characters is not required. If it is impossible to balance the string, return -1.

Example

brackets = ")()(())("

Swap the characters at the first and last index to get *"(()(()))"* which is balanced. The string can be balanced with1 swap.

Function Description

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

string brackets: the string to analyze

Returns

int: the minimum number of swaps or -1

Constraints

- 1 ≤ length of the string *brackets* ≤ 10^5
- *brackets* consists of *')'* and *'('* only.

▼ Input Format For Custom Testing

The first line contains a string, *brackets*.

▼ Sample Case 0

Sample Input For Custom Testing

```
(()))(
```

Sample Output

```
1
```

Explanation

Swap the last two brackets to get a balanced string *'(())()'*.

▼ Sample Case 1

Sample Input For Custom Testing

```
()())
```

Sample Output

```
-1
```

Explanation

The sequence can never be balanced.

Question - 26

Running Stream Median

The median is the middle value in a sorted list of numbers. For an even number of elements, the median is the $(n/2)^{th}$ element.

Given a continuous stream of integers, design a data structure that calculates the median of all elements seen so far each time a new integer is added to the stream.

Example

arr = [8, 2, 11, 9, 5]

The sorted list after each insertion is [8], [2, 8], [2, 8, 11], [2, 8, 9, 11], [2, 5, 8, 9, 11]. The medians after each insertion are 8, 2, 8, 8, and 8. Return [8, 2, 8, 8, 8].

Function Description

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

int arr[n]: an array of integers

Returns

int[n]: a record of medians of the running stream of numbers

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$

▼ 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 $0 \le i < n$) contains an integer, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
```

```
5         =>    arr[] size n = 5
2         =>    arr = [2, 4, 6, 1, 3]
4
6
1
3
```

Sample Output

```
2
2
4
2
3
```

Explanation

The numbers in ascending order after each integer is read are: [2], [2, 4], [2, 4, 6], [1, 2, 4, 6] and [1, 2, 3, 4, 6]. Thus, their respective medians are [2, 2, 4, 2, 3].

▼ Sample Case 1

Sample Input For Custom Testing

```
6
8
2
1
4
14
4
```

Sample Output

```
8
2
2
2
4
4
```

Explanation

The numbers in ascending order after each integer is read are: [8], [2, 8], [1, 2, 8], [1, 2, 4, 8], [1, 2, 4, 8, 14] and [1, 2, 4, 4, 8, 14]. Their respective medians are [8, 2, 2, 2, 4, 4].

Question - 27

Backspace String Compare

Two strings are considered identical if they have the same length and identical characters at each position. In a string, a backspace character '#' deletes the character immediately before it.

Given two strings containing lowercase English letters and the '#' character (representing a backspace), determine whether the resultant strings are identical after processing the backspaces. Return 1 if they are identical and 0 if they are not. Note that applying a backspace to an empty string leaves it unchanged.

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". The second string also becomes "axbc". The answer is 1.

Function Description

Complete the function *compareStrings* in the editor with 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* ≤ $2*10^5$
- 1 ≤ length of *s2* ≤ $2*10^5$
- 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 strings 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".

Question - 28

River Records

A meteorologist maintains a record of water level readings taken on a nearby river. One of the figures to determine is the maximum height above a previously recorded value that has been achieved to date. Given an array of integers, find the maximum difference between any element and any preceding smaller element without changing the order. If there is no such preceding element, return -1.

Example

arr = [5, 3, 6, 7, 4]

There are no earlier elements than *arr[0]*.
There is no earlier reading with a value lower than *arr[1].*
There are two lower earlier readings with a value lower than *arr[2] = 6:*

- *arr[2] - arr[1] = 6 - 3 = 3*
- *arr[2] - arr[0] = 6 - 5 = 1*

There are three lower earlier readings with a lower value than *arr[3] = 7:*

- *arr[3] - arr[2] = 7 - 6 = 1*
- *arr[3] - arr[1] = 7 - 3 = 4*

- *arr[3] - arr[0] = 7 - 5 = 2*

There is one lower earlier reading with a lower value than *arr[4] = 4:*

- *arr[4] - arr[1] = 4 - 3 = 1*

The maximum trailing record is *arr[3] - arr[1] = 4*.

Example

arr = [4, 3, 2, 1]

No item in *arr* has a lower earlier reading, therefore return *-1*

Function Description

Complete the function *maximumTrailing* in the editor below.

maximumTrailing has the following parameter(s):

int arr[n]: an array of integers

Returns:

int: the maximum trailing difference, or *-1* if no element in *arr* has a lower earlier value

Constraints

- $1 \le n \le 2 \times 10^5$
- $-10^6 \le arr[i] \le 10^6$ and $0 \le i < n$

▼ Input Format For Custom Testing

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

The first line contains a single integer, *n*, the number of elements in the array *arr*.
Each of the *n* subsequent lines contains a single integer, each an element *arr[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
7       →      arr[] size n = 7
2       →      arr = [2, 3, 10, 2, 4, 8, 1]
3
10
2
4
8
1
```

Sample Output

```
8
```

Explanation

Differences are calculated as:

- *3 - [2] = [1]*
- *10 - [3, 2] = [7, 8]*
- *4 - [2, 3, 2] = [2, 1, 2]*
- *8 - [4, 2, 3, 2] = [4, 6, 5, 6]*

The maximum trailing difference is *10 - 2 = 8*.

▼ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
6       →      arr[] size n = 6
7       →      arr = [7, 9, 5, 6, 3, 2]
9
5
6
3
2
```

Sample Output

```
2
```

Explanation

Differences are calculated as:

- *9 - [7] = 2*
- *6 - [5] = 1*

The maximum trailing difference is *2*.

Question - 29
The Coder Friends

Two participants, Erica and Bob, compete in a friendly hackathon where each solves one question per day. Each day features three questions of different difficulty levels: Easy (E), Medium (M), and Hard (H).

Points are awarded based on the difficulty of the solved question according to this scoring table:

Scoring table

Difficulty	Points
Easy (**E**)	1
Medium (**M**)	3
Hard (**H**)	5

You are given two strings, *erica* and *bob*, where each character represents the difficulty level of the question solved by the respective participant on each day. Calculate the total score for both participants and determine the winner: "Erica", "Bob", or "Tie" if they have equal scores.

Example

erica=["E"]
bob=["E"]

- Erica solved an Easy question worth 1 point.
- Bob solved an Easy question worth 1 point.
- Erica's and Bob's scores are equal: "Tie".

Function Description

Complete the function *winner* in the editor wiith the following parameter(s):

string erica[n]: erica[i] denotes the difficulty of Erica's problem solved on day *i*
string bob[n]: bob[i] denotes the difficulty of Bob's problem solved on day *i*

Returns

string: the winner's name, "Erica" or "Bob", or "Tie" if their scores are equal.

Constraints

- $0 < n < 10^5$

▼ Input Format For Custom Testing

The first line contains a string, *erica*.
The next line contains a string, *bob*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
EHH  →      erica=["E","H","H"]
EME  →      bob=["E","M","E"]
```

Sample Output

```
Erica
```

Explanation

```
               Erica's         Erica's      Bob's            Bob's
Day            difficulty      Score        difficulty       Score
0              E               1            E                1
1              H               5            M                3
2              H               5            E                1
Total Scores                   11                            5
```

Erica wins.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
E     →     erica=["E']
H     →     bob=["H"]
```

Sample Output

```
Bob
```

Explanation

```
               Erica's       Erica's      Bob's        Bob's
Day            difficulty    Score        difficulty   Score
0              E             1            H            5
Total Scores                 1                         5
```

Bob wins.

Question - 30
Count Binary Substrings

Count the number of substrings in a binary string that contain an equal number of 0s and 1s, where all 0s and 1s are grouped together. Duplicate substrings are counted in the total.

A binary string consists only of 0s and 1s, and a substring is a contiguous group of characters within the string.

Example:
s = "011001"

The qualifying substrings are "01", "10", "1100", and "01", giving a total of 4.
Note that "0110" has equal 0s and 1s but is not counted because the 0s and 1s are not grouped together.

Function Description
Complete the function *getSubstringCount* in the editor withthe following parameter(s):
s: a binary string

Returns
int: the number of substrings that meet the criteria

Constraints
- 1 ≤ length of s ≤ 10^5
- The string *s* consists of 0s and 1s only.

▼ Input Format For Custom Testing

The first line contains a string, s, a binary string.

▼ Sample Case 0

Sample Input For Custom Testing

```
001100011
```

Sample Output

```
6
```

Explanation
The substrings "01", "0011", "10", "1100", "01", and "0011" have an equal number of 0s and 1s with consecutive groupings.

▼ Sample Case 1

Sample Input For Custom Testing

```
000110
```

Sample Output

```
3
```

Explanation
The substrings "01", "0011", and "10" satisfy the constraints.

Question - 31
Zig-Zag Array

A zig-zag array alternates between elements being less than and greater than their adjacent elements. Transform an array of integers into a zig-zag array with the minimum number of replacements.

You can replace any element with any integer (positive, negative, or zero). Your task is to determine the minimum number of replacements needed.

Example
arr = [1, 2, 3, 4, 5]

There are two possible patterns:

1. LHLHL pattern (starting with a lower value):
   - [1, 2, ?, 4, ?]
   - Need to replace the 3 and 5 (2 replacements)
2. HLHLH pattern (starting with a higher value):
   - [?, 2, 3, ?, 5]
   - Need to replace the 1 and 4 (2 replacements)

Both patterns require 2 replacements, so the answer is 2.

Function Description

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

int *arr[n]:* an array of integers

Returns

int: the minimum number of operations required to turn *arr* into a zig-zag array

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$

▼ 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 $0 \le i < n$) contains an integer, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
8       →   n = 8
2       →   arr = [2, 1, 2, 3, 4, 5, 2, 9]
1
2
3
4
5
2
9
```

Sample Output

```
2
```

Explanation

```
Original: [2, 1, 2, 3, 4, 5, 2, 9]
           L  H  L  H  L  H  L  H
LHLHLHLH: [2, +, 2, 3, -, 5, 2, 9]

           H  L  H  L  H  L  H  H
HLHLHLHL: [2, 1, 2, -, 4, -, 2, -]
```

For the LHLH... pattern, replace the second value (1) with a number greater than 2 and the fifth value (4) with a number less than 3.
For the HLHL... pattern, replace the fourth value (3) with a number less than 2, and the sixth value (5) and the eighth value (9) with a number less than 2.
The LHLH... pattern only requires two replacements.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
6       →   arr[] size n = 6
1       →   arr = [1, 2, 4, 4, 5, 6]
```

```
2
4
4
5
6
```

Sample Output

```
2
```

Explanation

```
Original: [1, 2, 4, 4, 5, 6]
           L  H  L  H  L  H
LHLHLHLH: [1, 2, -, 4, -, 6]

           H  L  H  L  H  L
HLHLHLHL: [+, 2, 4, -, 5, -]
```

Starting with a low value takes 2 replacements, while starting with a high value takes 3. Return 2.

Question - 32
Sequence Construction

Construct a sequence of specified length using integers within a given range. The sequence must be strictly increasing and then strictly decreasing. The goal is to maximize the sequence elements from the beginning.

A sequence *[a, b, c, ...]* beats another sequence *[x, y, z, ...]* if:

- $a > x$, or
- $a = x$ and $b > y$, or
- $a = x$, $b = y$, and $c > z$, and so on

Given the length of the sequence and the range of integers, return the winning sequence or [-1] if no valid sequence is possible.

Example

$n = 5$

$lo = 3$

$hi = 10$

The winning sequence is [9, 10, 9, 8, 7].

This sequence is optimal because:

1. It follows the required pattern (increasing then decreasing)
2. It maximizes the first elements (starting with 9, then 10)
3. No other sequence with integers between 3 and 10 can beat it

Function Description

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

int n: the size of the sequence to create

int lo: the lower end of the integer range

int hi: the upper end of the integer range

Returns

int[n] or int[1]: the winning sequence of *n* integers, or *[-1]* if this is impossible

Constraints

- $3 \le n \le 10^5$
- $1 \le lo \le hi \le 10^5$

▼ Input Format For Custom Testing

The first line of input contains an integer, *n*.
The second line contains an integer, *lo*.
The third line contains an integer, *hi*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
5       →   n = 5
4       →   lo = 4
11      →   hi = 11
```

Sample Output

```
10
11
10
9
8
```

Explanation

Since 11 is the upper bound, a sequence starting with 11 cannot be strictly increasing at the first. The next best answer is the sequence starting with 10, which is [10, 11, 10, 9, 8].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
5       →   n = 5
1       →   lo = 1
2       →   hi = 2
```

Sample Output

```
-1
```

Explanation

It is impossible to construct a valid sequence using only 1 and 2. The only valid sequence with those two limits is [1, 2, 1]. Anything shorter or longer cannot be strictly increasing then strictly decreasing. Therefore, the answer is [-1].

Question - 33

Table of Contents

Design a system to extract a Table of Contents (TOC) from a document written in a simple markup format. The TOC should be structured based on the following rules:

1. Chapter Titles:
   - A line that begins with a single # followed by a space (# Title) represents a chapter.
2. Section Titles:
   - A line that begins with a double ## followed by a space (## Title) represents a section within a chapter.

The output should list chapter titles as top-level entries and section titles as indented sub-entries under their respective chapters.

For example, the input text is *n* = 12 lines long, where the *text* is the following:

```
# Algorithms
This chapter covers the most basic algorithms.
```

```
## Sorting
Quicksort is fast and widely used in practice
Merge sort is a deterministic algorithm
## Searching
DFS and BFS are widely used graph searching algorithms
Some variants of DFS are also used in game theory applications
# Data Structures
This chapter is all about data structures
It's a draft for now and will contain more sections in the future
# Binary Search Trees
```

This is the table of contents that must be produced:

```
1. Algorithms
1.1. Sorting
1.2. Searching
2. Data Structures
3. Binary Search Trees
```

Note that each number is followed by a period and the last period is followed by 1 space.

Function Description

Complete the function *tableOfContents* in the editor with the following parameter:

string text[n]: the lines in the table of contents

Returns

string[]: each string is a line in the table of contents

Constraints

- $1 \le n \le 1000$
- $1 \le$ length of *text[i]* ≤ 100
- When a line starts with # or with ##, these special characters are always followed by a space.
- The first line of the text is guaranteed to be a chapter line.

▼ Input Format for Custom Testing

In the first line, there is a single integer, *n*, the number of lines in *text*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string, *text[i]*.

▼ Sample Case 0

Sample Input

```
10
# Cars
Cars came into global use during the 20th century
Most definitions of car say they run primarily on roads
## Sedan
Sedan's first recorded use as a name for a car body was in 1912
## Coupe
A coupe is a passenger car with a sloping rear roofline and generally two doors
## SUV
The predecessors to SUVs date back to military and low-volume models from the late 1930s
There is no commonly agreed definition of an SUV, and usage varies between countries.
```

Sample Output

```
1. Cars
1.1. Sedan
1.2. Coupe
1.3. SUV
```

Explanation

The first input line indicates *n* = 10 lines of text. There is only 1 chapter in the input, containing 3 sections. All the lines that don't begin with # or ## are ignored in the table of contents.

▼ Sample Case 1

Sample Input

```
10
# Games
## Board
## Computer
## Zero sum
## Multiplayer
# Strategies
## Greedy
## Tree pruning
## Others
# Summary
```

Sample Output

```
1. Games
1.1. Board
1.2. Computer
1.3. Zero sum
1.4. Multiplayer
2. Strategies
2.1. Greedy
2.2. Tree pruning
2.3. Others
3. Summary
```

Explanation

Again, the first input line indicates there are *n* = 10 lines of text. This text already looks like an outline because it contains only chapters and sections. Chapter 1 has 4 sections in it, Chapter 2 has 3 sections, and Chapter 3 has no sections.

Question - 34

Smallest Negative Balance

You are working on a new application for recording debts. This program allows users to create groups that show all records of debts between the group members. Given the group debt records (including the borrower name, lender name, and debt amount), who in the group has the smallest negative balance?

Notes:

- *-10* is smaller than *-1*
- If multiple people have the smallest negative balance, return the list in alphabetical order.
- If nobody has a negative balance, return the string array ["Nobody has a negative balance"].

Example

n = 6

debts = ['Alex Blake 2', 'Blake Alex 2', 'Casey Alex 5', 'Blake Casey 7', 'Alex Blake 4', 'Alex Casey 4']

There are 6 debt records, as shown in the table below:

borrower		lender	amount
Alex	Blake	2	
Blake	Alex	2	

Casey	Alex	5
Blake	Casey	7
Alex	Blake	4
Alex	Casey	4

- The first, fifth, and sixth entries decrease Alex's balance because Alex is a borrower. The second and third entries increase it because Alex is a lender. Alex's balance is (2+5) - (2+4+4) = 7 - 10 = -3.
- Blake is a lender in the first and fifth entries and a borrower in the second and fourth entries. Thus, Blake's balance is (2+4) - (2+7) = 6 - 9 = -3.
- Casey is a lender in the fourth and sixth entries, and a borrower in the third entry. Thus, Casey's balance is (7+4) - 5 = 6.

The answer (in alphabetical order) is ["Alex", "Blake"] because both of them have a balance of -3, which is the minimum among all members.

Function Description

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

int *debts[n][3]*: a 2-dimensional array of strings, where *debts[i][0]* denotes the borrower, *debts[i][1]* denotes the lender, and *debts[i][2]* denotes the debt amount

Returns

string[]: the alphabetically ordered list of members with the smallest negative balance, or an array containing the string "Nobody has a negative balance"

Constraints

- $1 \le n \le 2 \times 10^5$
- *debts[i][2]* represents an integer between 1 and 1000 inclusively.
- 1 ≤ length of *debts[i][0]*, length of *debts[i][1]* ≤ 20
- The first character of *debts[i][0]* and *debts[i][1]* is a capital English letter.
- Every character of *debts[i][j]* and *debts[i][1]* except the first one is a lowercase English letter.
- *debts[i][0] ≠ debts[i][1]*.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of rows in *debts*.

The second line contains an integer, 3, the number of elements in *debts[i]*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains *3* space-separated strings: *debts[i][0], debts[i][1], and debts[i][2]*, borrower, lender, amount.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN              Function
-----              --------
5            →     debts[] size n = 5
3            →     debts[i] size = 3 (always)
Alex Blake 5 →     debts = [['Alex', 'Blake', '5'], ['Blake', 'Alex', '3'], ['Casey', 'Alex', '7'],\
Blake Alex 3                ['Casey', 'Alex', '4'], ['Casey', 'Alex', '2']]
Casey Alex 7
Casey Alex 4
Casey Alex 2
```

Sample Output

```
Casey
```

Explanation

borrower	lender	amount

Alex	Blake	5
Blake	Alex	3
Casey	Alex	7
Casey	Alex	4
Casey	Alex	2

- Alex lends (3+7+4+2) and borrows 5, so the balance is 16 - 5 = 11.
- Blake lends 5 and borrows 3, so the balance is 5 - 3 = 2.
- Casey lends 0 and borrows (7+4+2), so the balance is 0 - 13 = -13.

The answer is ["Casey"] because Casey has the smallest (and only) negative balance.

▼ Sample Case 1

Sample Input For Custom Testing

```
5
3
Blake Alex 7
Blake Alex 3
Alex Blake 4
Blake Alex 1
Alex Blake 7
```

Sample Output

```
Nobody has a negative balance
```

Explanation

borrower		lender	amount
Blake	Alex	7	
Blake	Alex	3	
Alex	Blake	4	
Blake	Alex	1	
Alex	Blake	7	

- Alex lends (7+3+1) and borrows (4+7), so the balance is 11 - 11 = 0.
- Blake lends (4+7) and borrows (7+3+1), so the balance is 11 - 11 = 0.

Since nobody has a negative balance, the answer is ["Nobody has a negative balance"].

Question - 35
Minimizing a String

Transform a string into the alphabetically smallest string possible by repeatedly swapping adjacent 'a' and 'b' characters or adjacent 'b' and 'c' characters.

Note: A string *x* is alphabetically smaller than a string *y* if, for the first index *i* where *x* and *y* differ, *x[i] < y[i]* .

Example

s = "abaacbac"

The alphabetically smallest possible string is obtained by applying the following operations:

- *'c'* at index 4 is swapped with '*b'* at index 5. So "*abaacbac"* becomes "*abaabcac"*.
- Then, '*b'* at index 1 is swapped with '*a'* at index 2. So "*abaabcac"* becomes "*aababcac"*.
- Finally, '*b'* at index 2 is swapped with '*a* 'at index 3 to obtain the final answer: "*aaabbcac".*

Function Description

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

string s: the string to minimize

Returns

string: the alphabetically smallest string obtained after swapping

Constraints

- $1 \leq$ length of $s \leq 10^5$
- *s* only contains the characters '*a*', '*b*', and '*c*'.

▼ Input Format For Custom Testing

The only line contains a string, *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
baacba  →   s = "baacba"
```

Sample Output

```
aabbca
```

Explanation

Swap '*c'* at index 4 with '*b'* at index 3 to get "*baa**bc**a"*.
Swap '*b'* at index 0 with '*a'* at index 1 and then with the '*a'* at index 2 to get "***aab**bca"*.
This is the alphabetically smallest string possible.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         Function
-----         --------
ababbaab    → s = "ababbaab"
```

Sample Output

```
aaaabbbb
```

Explanation

Swap '*b'* at index 4 with '*a'* at index 5 and then with '*a'* at index 6 to get "*abab**aab**b"*. Follow a similar procedure for the '*b*'s at index 3 and 1 to get the final answer, "***aaaabb**bb"*.

Question - 36
Process Scheduling

A multiprocessor system has a certain number of processors, each with the ability to schedule a limited number of processes in one second. After scheduling, a processor's ability is reduced to *floor(ability/2)*.

Given the processors' abilities and the number of processes, determine the minimum time required to schedule all processes.

Example

n = 5

ability = [3, 1, 7, 2, 4]

processes = 15

1. Use the processor with *ability* = 7 to schedule 7 processes. Ability becomes floor(7/2) = 3. Remaining processes = 8.
2. Use the processor with *ability* = 4 to schedule 4 processes. Ability becomes floor(4/2) = 2. Remaining processes = 4.
3. Use a processor with *ability* = 3 to schedule 3 processes. Ability becomes floor(3/2) = 1. Remaining process = 1.
4. Use a processor with *ability* = 1 to schedule the final process.

Each step requires one second, so the answer is 4 seconds.

Function Description

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

int ability[n]: each element denotes the ability of the i^{th} processor

long processes: the number of processes to schedule

Returns

int: the minimum time required to schedule the processes

Constraints

- $1 \le n \le 10^5$
- $1 \le ability[i] \le 10^6$
- $1 \le processes \le 10^{12}$
- It is guaranteed that the processes can be scheduled using the given multiprocessor system.

▼ Input Format For Custom Testing

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

Each of the next n lines (where $0 \le i < n$) contains an integer, *ability[i].*

The next line contains an integer, *processes*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
5       →   number of processors n = 5
2       →   ability = [2, 1, 5, 3, 1]
1
5
3
1
17      →   processes = 17
```

Sample Output

```
9
```

Explanation

After choosing the processors with *ability* = 2, 5, and 3, respectively, a total of 10 processes are scheduled (requiring 3 seconds, one for each processor). So, there are 7 remaining processes, and the updated *ability* array now becomes [1, 1, 2, 1, 1].

Then choose the processor with *ability* = 2 to schedule 2 processes (requiring 1 second). After that, *ability* = [1, 1, 1, 1, 1] and 5 processes remain. All 5 processors are used to schedule one process each (requiring 5 seconds).

The total time required is (3+1+5) = 9.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
----------     -----------
4     → number of processors n = 4
3     →    ability = [3, 1, 4, 2]
1
4
2
3     →    processes = 3
```

Sample Output

```
1
```

Explanation

Choose the processor with *ability* = 3. All 3 processes are scheduled, and the total time required is 1 second.

Question - 37

Suspicious Activity From Logs

Application logs provide valuable insights into user interactions and can help detect suspicious activities.

You are given a log file represented as a string array, where each entry records a money transfer in the following format: "sender_user_id recipient_user_id amount". Each entry consists of:

- sender_user_id: The user initiating the transfer.
- recipient_user_id: The user receiving the transfer.
- amount: The transferred amount.

The sender_user_id, recipient_user_id, and amount are numeric (0-9), up to 9 digits long, and cannot start with zero. The log entries are provided in no particular order.

Write a function that identifies suspicious users–users who appear in at least *threshold* number of log entries, whether as a sender or a recipient. Return an array of user IDs that meet the threshold condition, sorted in ascending numerical order.

Example

logs = ["88 99 200", "88 99 300", "99 32 100", " 12 12 15"]
threshold = 2

The transactions count for each user, regardless of role, are:

```
ID      Transactions
-----      ----------
99      3
88      2
12      1
32      1
```

There are two users with at least *threshold* = 2 transactions: 99 and 88. The returned array is ["88", "99"] in ascending order.

Note: In the last log entry, user 12 was on both sides of the transaction. This counts as only 1 transaction for user 12.

Function Description

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

string logs[n]: each *logs[i]* denotes the i^{th} entry in the logs.

int threshold: the minimum number of transactions in which a user must be involved (as either sender or recipient) to be included in the result.

Returns

string[]: A sorted array of user IDs that appear in at least the *threshold* number of transactions.

Constraints

- $1 \le n \le 10^5$
- $1 \le threshold \le n$
- The *sender_user_id, recipient_user_id,* and *amount* contain only characters in the range ascii['0'-'9'].
- The *sender_user_id, recipient_user_id,* and *amount* start with a non-zero digit.
- $0 <$ length of *sender_user_id, recipient_user_id, amount* ≤ 9.
- The result will contain at least one element.

▼ Input Format for Custom Testing

The first line contains the integer, *n,* the size of *logs*.
The following *n* lines contain a string, *logs[i]*.
The last line contains an integer, *threshold.*

▼ Sample Case 0

Sample Input

```
STDIN     Function
-------   ------------
4       → logs[] size n = 4
1 2 50 → logs = ["1 2 50", "1 7 70", "1 3 20", "2 2 17"]
1 7 70
1 3 20
2 2 17
2       → threshold = 2
```

Sample Output

```
1
2
```

Explanation

```
ID      Transactions
-----   ------------
1       3
2       2
7       1
3       1
```

Only users *1* and *2* have at least *threshold* = 2 transactions. The returned array in numerically ascending order is ["1", "2"]. Note that in the last log entry, the user with ID 2 performed both roles in the transaction, which is counted as one transaction for the user.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-------   ------------
4       → logs[] size n = 4
9 7 50 → logs = ["9 7 50", "22 7 20", "33 7 50", "22 7 30"]
22 7 20
33 7 50
22 7 30
3       → threshold = 3
```

Sample Output

```
7
```

Explanation

```
ID      Transactions
-----   ------------
9       1
7       4
22      2
33      1
```

Only user 7 has *3* or more transactions. The returned array is ["7"].

Question - 38
Minimum Processing Time

Optimize task assignment across multiple processors to minimize completion time. A computing cluster has multiple processors, each with 4 cores. The number of tasks equals the total number of cores in the cluster.

Each task has a predicted execution time, *taskTime[i]*, and each processor has a specified time when its cores become available, *processorTime[i]*. Exactly 4 tasks must be assigned to each processor, and tasks run independently on the cores.

Determine the earliest possible time when all tasks can be completed.

Example

n = 2 processors
processorTime = [8, 10]
taskTime = [2, 2, 3, 1, 8, 7, 4, 5]

An optimal solution assigns:

- Tasks with times 2, 3, 7, and 8 to processor 0 (available at time 8)
- Tasks with times 4, 2, 5, and 1 to processor 1 (available at time 10)

Completion times:

- Processor 0: 10, 11, 15, and 16
- Processor 1: 14, 12, 15, and 11

The earliest possible finish time is 16.

Function Description

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

int processorTime[n]: the times when each processor becomes available
int taskTime[4×n]: task execution times

Returns

int: the earliest time the tasks can be completed

Constraints

- $1 \le n \le 10^5$
- $1 \le processorTime[i] \le 10^6$
- $1 \le taskTime[i] \le 10^6$

▼ Input Format For Custom Testing

The first line contains an integer, *n,* the number of processors.
Each of the next *n* lines contains an integer, *processorTime[i]*.
The next line contains an integer, $(4 \times n)$, the number of tasks.
Each of the next $(4 \times n)$ lines contains an integer, *taskTime[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
2       →   number of processors, n = 2
10      →   processorTime = [10, 20]
20
8       →   number of tasks = 8
2       →   taskTime = [2, 3, 1, 2, 5, 8, 4, 3]
3
1
2
5
8
4
3
```

Sample Output

```
23
```

Explanation

There are 2 processors that become available at times 10 and 20, and there are 8 tasks with execution times 2, 3, 1, 2, 5, 8, 4, and 3. It is optimal to do the following:

- Assign to the first processor (which becomes available at time 10) the tasks with the following execution times: 1, 4, 5, and 8.
- Assign to the second processor (which becomes available at time 20) the tasks with the following execution times: 2, 2, 3, and 3.

The tasks assigned to the first processor finish at times (10+1), (10+4), (10+5), and (10+8), which are 11, 14, 15, and 18, respectively.
The tasks assigned to the second processor finish at times (20+2), (20+2), (20+3), and (20+3), which are 22, 22, 23, and 23, respectively.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
1       →   number of processors n = 1
10      →   processorTime = [10]
4       →   number of tasks = 4
10      →   taskTime = [10, 10, 10, 10]
10
10
10
```

Sample Output

```
20
```

Explanation

There is only one processor, and it becomes available at time 10. There are 4 tasks, all with execution times of 10. The only possible assignment is to assign all the tasks to the only processor. Since all of them have the same execution time, they will all finish at time 10+10 = 20.

Question - 39
Sign-in Sign-out Logs

Application logs are analyzed to understand interactions with an application and can be used to detect specific actions.

You are provided with a log file in the form of a string array where each entry is formatted as "*user_id timestamp action*", with each value separated by a space.

- Both *user_id* and *timestamp* are composed only of digits, are a maximum of 9 digits long, and start with a non-zero digit.
- The *timestamp* indicates the time in seconds since the application was last launched.

- The action will either be "sign-in" or "sign-out".

Given a log with entries in no particular order, return an array of strings representing IDs of users who signed out within *maxSpan* seconds of signing in, sorted in ascending numerical order.

Example

n = 7

logs = ["30 99 sign-in", "30 105 sign-out", "12 100 sign-in", "20 80 sign-in", "12 120 sign-out", "20 101 sign-out", "21 110 sign-in"]

maxSpan = 20

```
ID   Sign in   Sign out   Time delta
---  --------  ---------  ----------
30   99        105        6
12   100       120        20
20   80        101        21
21   110
```

The users with IDs 30 and 12 were not signed in for more than *maxSpan* = 20 seconds. The return array is ["12", "30"] in sorted numerical order.

Function Description

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

string logs[n]: each *logs[i]* denotes the i^{th} entry in the logs

int maxSpan: the maximum difference in seconds between when a user logs in and logs out for the user to be included in the result

Returns

string[]: a string array of user IDs, sorted ascending by numeric value

Constraints

- $1 \le n, maxSpan \le 10^5$
- $1 \le maxSpan \le n$
- Each *user_id* contains only digits '0'-'9'], is at most *9* digits long, and starts with a non-zero digit.
- Each *timestamp* contains only digits and begins with a non-zero digit
- $0 < timestamp \le 10^9$
- Each *action* is either "sign-in" or "sign-out".
- Each *user_id*'s *sign-in timestamp* < *sign-out timestamp*
- Each user signs in for only 1 session.
- The results will contain at least one element.

▼ Input Format Format for Custom Testing

The first line contains an integer, *n,* the size of *logs*.
Each of the next *n* lines contains a log file entry, *logs[i].*
The last line contains a single integer, *maxSpan*.

▼ Sample Case 0

Sample Input

```
STDIN                Function
-----                --------
6              →     logs[] size n = 6
99 1 sign-in   →     logs = ["99 1 sign-in","100 10 sign-in","50 20 sign-in","100 15 sign-out","50 26 sign-
out","99 2 sign-out"]
100 10 sign-in
50 20 sign-in
100 15 sign-out
```

```
50 26 sign-out
99 2 sign-out
5               →    maxSpan = 5
```

Sample Output

```
99
100
```

Explanation

```
ID  Sign in  Sign out  Time delta
---  ----------  ----------  ----------
50  20        26          6
99  1         2           1
100 10        15          5
```

The users with IDs 99 and 100 were not signed in for more than *maxSpan = 5* seconds. In sorted numerical order, the return array is ["99", "100"].

▼ Sample Case 1

Sample Input

```
STDIN               Function
-----               --------
4                →  logs[] size n = 4
60 12 sign-in    →  logs = ["60 12 sign-in","80 20 sign-out","10 20 sign-in","60 20 sign-out"]
80 20 sign-out
10 20 sign-in
60 20 sign-out
100              →  maxSpan = 100
```

Sample Output

```
60
```

Explanation

```
ID  Sign in  Sign out  Time delta
-----  ----------
10  20
60  12         20         8
80  20
```

Only user ID *60* has signed out and was not signed in for more than *maxSpan = 100* seconds. The return array is ["60"].

Question - 40

Area of Triangle (Easy)

Given three sets of distinct coordinates that form a triangle, calculate the area of the triangle. At least one side of the triangle will be parallel to either the x-axis or the y-axis.

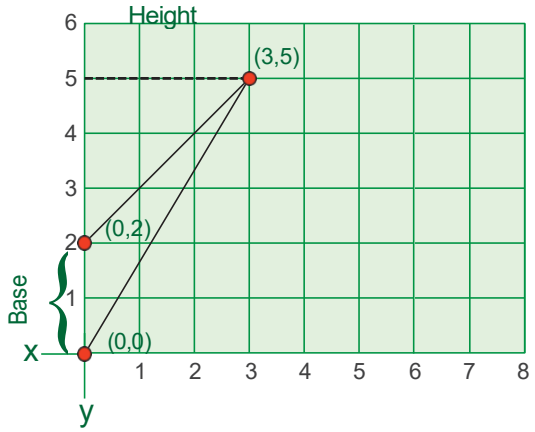
Example

x = [0, 3, 0]

y = [0, 5, 2]

Aligned by index, the 3 coordinates are [0,0], [3,5], [0,2]. The base of the triangle is 2, and the height is 3. The area of a triangle is (*base* * *height*)/2, so 3 × 2 / 2 = 3.

All resulting areas will be whole numbers.



Function Description

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

int x [3]: the *x* coordinates

int *y[3]:* the *y* coordinates, aligned with *x* by index

Returns

long int: the area of the triangle

Constraints

- $0 \le x[i], y[i] < 10^5$

▼ Input Format For Custom Testing

The first line gives the number of *x* coordinates (always 3).
The next 3 lines each contain a single integer, *x[i]*.
The 4th line contains the number of *y* coordinates (always 3).
The next 3 lines each contain a single integer, *y[i]*.

▼ Sample Case 0

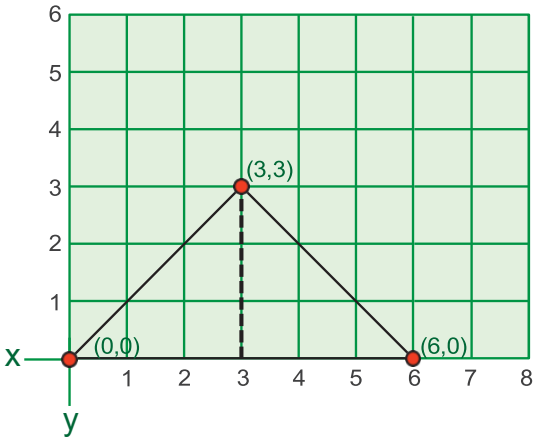
Sample Input 0

```
STDIN       Function
-----       --------
3       →   x[] size = 3 (always)
0       →   x = [0, 3, 6]
3
6
3       →   y[] size = 3 (always)
0       →   y = [0, 3, 0]
3
0
```

Sample Output 0

```
9
```

Explanation 0



The base has a length of 6, and the height is 3. The area is (6 * 3) / 2 = 9.

▼ Sample Case 1

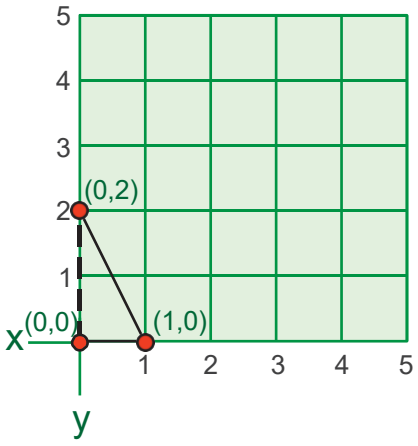
Sample Input 1

```
STDIN       Function
-----       --------
3     →     x[] size = 3
0     →     y = [0, 1, 0]
1
0
3     →     y[] size = 3
0     →     y = [0, 0, 2]
0
2
```

Sample Output 1

```
1
```

Explanation 1



The base of the triangle is 1 and the height is 2, so the area is (1 * 2) / 2 = 1.

Question - 41
How Many Words

A word is defined as a sequence of letters ('a'-'z', A-Z) that may contain hyphens and may end with punctuation marks (period '.', comma ',', question mark '?', exclamation point '!"). Words are separated by one or more whitespace characters. Hyphens join two words into one and should be retained, while other punctuation marks should be stripped before counting.

Count the number of valid words in a string, excluding numeric substrings.

Example
s = "How many eggs are in a half-dozen, 13?""

Return: 7

The words identified are: ["How", "many", "eggs", "are", "in", "a", "half-dozen"]

Note that the numeric string "13" is not counted as a word because it contains only digits, which are not in the allowed character set.

Function Description
Complete the function *howMany* in the editor with the following parameter(s):
sentence: a string

Returns:
int: the number of words in the string

Constraints
- 0 < length of $s \leq 10^5$

▼ Input Format For Custom Testing

The only line contains a string, *sentence*.

▼ Sample Case 0

Sample Input

```
he is a good programmer, he won 865 competitions, but sometimes he dont. What do you think? All test-cases should pass. Done-done?
```

Sample Output

```
21
```

Explanation
The substring "865" is not a word, so is not included in the count. The hyphenated words 'test-cases' and "Done-done" each count as 1 word.

▼ Sample Case 1

Sample Input

```
jds dsaf lkdf kdsa fkldsf, adsbf ldka ads? asd bfdal ds bf[l. akf dhj ds 878  dwa WE DE 7475 dsfh ds  RAMU 748 dj.
```

Sample Output

```
21
```

Explanation
Note that the substring "bf[l" is not a word because of the invalid character. Other substrings that are not words are "878", "7475" and "748".

Question - 42
Subarray MaxMin

Given an array, for each subarray of a specified length, find the smallest element in the subarray. Among all these smallest elements, determine the largest one.

The subarrays are formed by taking consecutive elements starting from each position in the array, with each subarray having the specified length. The last valid subarray ends exactly at the last element of the array.

Example

n = 5, the number of elements
arr = [1, 2, 3, 4, 5]
k = 2

For subarray size *k* = 2, the subarrays are [1, 2], [2, 3], [3, 4], and [4, 5], and their minima are [1, 2, 3, 4]. The final answer is *4*, the maximum of these.

Function Description

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

int arr[n]: an array of integers
int k: the subarray length

Returns

int: an integer indicating the maximum of the minima of the subarrays

Constraints

- $1 \le n \le 10^6$
- $1 \le arr[i] \le 10^9$ $(0 \le i < n)$
- $1 \le k \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr[]*.
Each line *i* of the next *n* lines contains an integer, *arr[i]*.
The last line contains an integer, *k*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
5       →   arr[] size n = 5
1       →   arr = [1, 2, 3, 1, 2]
2
3
1
2
1       →   k = 1
```

Sample Output

```
3
```

Explanation

Each element of *arr* = [1, 2, 3, 1, 2] is an array of size *k* = 1 and its minimum. The maximum of these minima is 3.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
3       →   arr[] size n = 3
1       →   arr = [1, 1, 1]
```

```
1
1
2       →  k = 2
```

Sample Output

```
1
```

Explanation
The two subarrays of size *k* = 2 are [1, 1], [1, 1]. Their minima are [1, 1], and the maximum of these is 1.

▼ Sample Case 2

Sample Input

```
STDIN       Function
----------      --------
5       →  arr[] size n = 5
2       →  arr = [2, 5, 4, 6, 8]
5
4
6
8
3       →  k = 3
```

Sample Output

```
4
```

Explanation
The subarrays of *arr* = [2, 5, 4, 6, 8] of size *k* = 3 are [2, 5, 4], [5, 4, 6], and [4, 6, 8]. Their minima are [2, 4, 4], and the maximum of these is 4.

Question - 43
Array Challenge

For each element in an array, implement a function that:

1. Initializes a counter to 0 for each element
2. Compares the element with each element to its left:
   - If the left element is greater, subtract the absolute difference from the counter
   - If the left element is smaller, add the absolute difference to the counter
3. Returns a new array containing the final counter values for each element

Example
n = 3, the number of elements
arr = [2, 4, 3]

- For *arr[0]* = 2, *counter* starts at 0, and there are no elements to the left, so the *counter* = *0*.
- For *arr[1]* = 4, *counter* starts at 0 and then increases by | 4 - 2 | = 2 at the first and only comparison: *counter* = 2.
- Testing *arr[2]* = 3, first against 4, *counter* = 0 - |3 - 4| = -1, and then against *2*, *counter* = -1 + |3 - 2| = 0.
- The answer array is [0, 2, 0].

Function Description
Complete the function *arrayChallenge* in the editor with the following parameter(s):
int arr[n]: an array of integers

Returns

int[n]: an array of integers calculated as described

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$

▼ 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 $0 \le i < n$) contains an integer that describes *arr[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     --------
3       →   arr[] size n = 3
2       →   arr = [2, 1, 3]
1
3
```

Sample Output

```
0
-1
3
```

Explanation

The first element has none to its left, so *counter* = 0.
The second element, *counter* = 0 - |1- 2| = -1.
The third element, *counter* = 0 + |3 - 1| + |3 - 2| = 3.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-------     --------
4       →   arr[] size n = 4
1       →   arr = [1, 2, 2, 3]
2
2
3
```

Sample Output

```
0
1
1
4
```

Explanation

The first element has none to its left, so *counter* = 0.
The second element, *counter* = 0 + |2 - 1| = 1.
The third element, *counter* = 0 + |2 - 2| + |2 - 1| = 1.
The fourth element, *counter* = 0 + |3 - 2| + |3 - 2| + |3 - 1| = 4.

Question - 44
Freelancing Platform

A client has posted multiple web development projects on a freelancing website, and different developers have submitted bids for these projects. Given the information about project bids, determine the minimum total cost for the client to complete all projects.

Return the minimum possible cost for completing all projects. If any project does not receive any bids, return -1.

Example

numProjects = 3 projects.
projectId = [2, 0, 1, 2]
bid = [8, 7, 6, 9].

- *projectId[i]* is aligned with *bid[i]*
- The first web developer bid 8 for project 2.
- The second web developer bid 7 for project 0.
- The third web developer bid 6 for project 1.
- The fourth web developer bid 9 for project 2.

There is only one choice of who to hire for project 0, and it will cost 7. Likewise, there is only one choice for project 1, which will cost 6. For project 2, it is optimal to hire the first web developer, instead of the fourth, and doing so will cost 8. So the final answer is 7 + 6 + 8 = 21.

If instead there were four projects, the answer would be -1 since there were no bids received on the fourth project.

Function Description

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

int numProjects: the total number of projects labeled from 0 to *numProjects - 1*

int projectId[n]: the projects that the freelancers bid on

int bid[n]: the bid amounts posted by the freelancers

Returns

long: the minimum cost the client can spend to complete all projects, or *-1* if any project has no bids.

Constraints

- $1 \le numProjects, n \le 5 \times 10^5$
- $0 \le projectId[i] < n$
- $1 \le bid[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *numProjects*.
The next line contains an integer, *n*, which denotes the number of elements in *projectId[]*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *projectId[i]*.
The next line contains an integer, *n*, which denotes the number of elements in *bid[]*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *bid[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
2       →   numProjects = 2
5       →   projectId[] size n = 5
0       →   projectId = [0, 1, 0, 1, 1]
1
0
1
1
5       →   bid[] size n = 5
4       →   bid = [4, 74, 47, 744, 7]
74
47
```

```
744
7
```

Sample Output

```
11
```

Explanation
The bids are as follows:

- The first web developer bid 4 for project 0.
- The second web developer bid 74 for project 1.
- The third web developer bid 47 for project 0.
- The fourth web developer bid 744 for project 1.
- The fifth web developer bid 7 for project 1.

The optimal solution is to hire the first web developer to complete project 0 (which costs 4) and to hire the fifth web developer to complete project 1 (which costs 7). This brings the total cost to 11.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-------     --------
2       →   numProjects = 2
2       →   projectId[] size n = 2
1       →   projectId = [1, 1]
1
2       →   bid[] size n = 2
4       →   bid = [4, 7]
7
```

Sample Output

```
-1
```

Explanation
Since there are no bids for project 0, the function should return -1.

Question - 45
Balancing Parentheses

Given a string composed of left and right parentheses, '(', and ')', balance the parentheses by inserting the necessary parantheses. Determine the minimum number of insertions required.

Example
s = '(()))'

Insert *1* left parenthesis at the left end of the string to get '((()))'. The string is balanced after *1* insertion.

s = '))(('

Insert *2* left parentheses at the start and *2* right parentheses at the end of the string to get '(())(())' after *4* insertions.

Function Description
Complete the function *getMinOperations* in the editor with the following parameter(s):

string s: a string of parentheses

Returns
int: the minimum number of insertions required to balance the parentheses

Constraints
- $1 \le$ length of $s \le 10^5$

▼ Input Format For Custom Testing

The first line contains a string, *s*, the initial parentheses sequence.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
())) →      s = '()))'
```

Sample Output

```
2
```

Explanation
Insert a '(' *2* times at the beginning of the string to make it valid: '(())'.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
()() →      s = '()()'
```

Sample Output

```
0
```

Explanation
The sequence is already valid, so no insertions are needed.

Question - 46
Binary Storage

A scientist needs to store DNA information in a database where each record is represented as a binary number with a fixed number of set bits. To optimize storage, these binary numbers are stored as lists of integers representing the indices of the set bits (1 bits), starting from the rightmost bit (least significant bit) and given in random order.

Given such a list of numbers, sort them in descending order based on their decimal values and return the sorted list of their original indices.

Example
bitArrays = [[0, 2], [2, 3], [2, 1]]

```
i    binary   decimal
0    0101       5
1    1100      12
2    0110       6
```

```
Return: [1, 2, 0]
```

The values sorted in descending order are [12, 6, 5]. The indices of those values are [1, 2, 0].

Function Description

Complete the function *sortBinaryNumbers* in the editor with the following parameters:
int bitArrays[n][m]: a 2D array of integers

Return

int[n]: an array of integers

Constraints

- $1 \le n, m \le 10^3$
- $0 \le bitArrays[i][j] \le 10^9$
- All integers in a row are distinct.
- Each row generates a unique binary number.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of binary numbers (size of *bitArrays)*.
The second line contains an integer, *m*, the number of set bits of each binary number (size of *bitArrays[i]*).

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

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
2     →     bitArrays[] size n = 2
3     →     bitArrays[i][] size m = 3
0 1 2 →     bitArrays = [[0, 1, 2], [3, 1, 0]]
3 1 0
```

Sample Output

```
1
0
```

Explanation

The corresponding numbers are:

index 0: $0111_2 = 7_{10}$
index 1: $1011_2 = 11_{10}$

The indices in order of the numbers ordered from largest to smallest are [1, 0]

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
---------  -----------
3    →     bitArrays[] size n = 3
1    →     bitArrays[i][] size m = 1
1    →     bitArrays = [[1], [2], [0]]
2
0
```

Sample Output

```
1
0
2
```

Explanation

The corresponding numbers are:

index 0: $010_2 = 2_{10}$
index 1: $100_2 = 4_{10}$
index 2: $001_2 = 1_{10}$

The indices in order of the numbers ordered from largest to smallest are [1, 0, 2]

Question - 47

Group Division

A university has admitted *n* students with varying skill levels. To create appropriate classes, the university needs to group students based on their skill levels. Each student has a skill level represented in the array *levels*. All students within a group must have skill levels within *maxSpread* of each other. Your task is to determine the minimum number of classes that must be formed.

Example

n = 5
levels = [1, 4, 7, 3, 4]
maxSpread = 2

- Students in any group must have skill levels within 2 of each other.
- Possible groupings:
  - Option 1: (1, 3), (4, 4), (7)
  - Option 2: (1), (3, 4, 4), (7)
- Both options require 3 groups.
- There is no way to form fewer than 3 groups.

The minimum number of classes required is 3.

Function Description

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

int levels[n]: the skill level for each student

int maxSpread: the maximum allowed skill level difference between any two members of a group

Returns

int: the minimum number of groups that can be formed

Constraints

- $1 \le n \le 10^5$
- $1 \le levels[i] \le 10^9$ for every i (where $0 \le i < n$)
- $0 \le maxSpread \le 10^9$

▼ Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *levels[i]*.
The next line contains an integer, *maxSpread*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4     →     levels[] size n = 4
4     →     levels[] = [ 4, 8, 1, 7 ]
8
1
7
3     →    maxSpread = 3
```

Sample Output

```
2
```

Explanation

The students are grouped into 2 groups as [1, 4] and [7, 8].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
5     →     levels[] size n = 5
4     →     levels[] = [ 4, 1, 2, 5, 3 ]
1
2
5
3
0     →    maxSpread = 0
```

Sample Output

```
5
```

Explanation

There are no students with matching levels so each student must be in a separate group.

Question - 48

How Many Flips?

Start with an initial string of zeros. Choose any digit to flip. When a digit is flipped, its value and those to the right switch state between 0 and 1. Given a target string of binary digits, determine the minimum number of flips required to achieve the target.

Example:

target = 01011

Start with a string of *5* zeros, the same length string as the *target.*

Initial String -> 00000

Flip the 3rd digit -> 00111

Flip the 2nd digit -> 01000

Flip the 4th digit -> 01011

3 flips are required to reach the target. The return value is *3.*

Function Description

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

string target: a string of *0*s and *1*s to match

Returns

int: the minimum number of flips needed to obtain the target string

Constraints

- $1 \le length\ of\ target \le 10^5$
- $0 \le target[i] \le 1$
- The *target* string consists of digits *0* and *1*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the length of the *target*.
The next line contains a string, *target*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
0011  →   target = '0011'
```

Sample Output

```
1
```

Explanation

The number of digits is *length(target) = 4*.
Flip the 3^{rd} digit to obtain the desired state: *0000 → 0011* after *1* flip.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
1010  →   target = '1010'
```

Sample Output

```
4
```

Explanation

Flip the 4^{th}, 3^{rd}, 2^{nd} and 1^{st} digits to produce *0000→ 0001→ 0010→ 0101→ 1010* after *4* operations.

Question - 49
Local Web Hosting

A web developer has set up a local web host on a low-power computer that also hosts a higher-priority media server. Each second that the web host has priority, it responds to at most 5 requests and removes them from the queue. Requests are served on a last-in-first-out basis.

Given a list of timestamps when requests are received, called *timestamp,* and a list of times when the web host has priority, called *top*, determine the total number of requests that are served.

Example
n = 6 requests received
timestamp = [0, 1, 1, 2, 4, 5]
m = 1 priority times
top = [5]

The web server has priority once at time 5. It can respond to the 5 most recent requests (those received at timestamps 1, 1, 2, 4, and 5). The request from timestamp 0 is ignored. The total number of requests served is 5.

Note:

- The server may have priority multiple times in a second.
- The arrays *timestamp* and *top* may not be in sorted order.

Function Description

Complete the function *requestsServed* in the editor below with the following parameters:

int timestamp[n]: the seconds after *time* = 0 that the requests arrive

int top[m]: the seconds after *time* = 0 that the server has priority

Returns

int: the number of served requests

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 30$
- $0 \le timestamp[i] < 60$
- $0 \le top[j] < 60$

▼ Input Format For Custom Testing

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

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

The next line contains an integer *m,* the size of *top*.

Each of the next *m* lines contains an integer, *top[j]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
10          timestamp[] size n = 10
1           timestamp = [1, 2, 2, 3, 4, 5, 6, 6, 13, 16]
2
2
3
4
5
6
6
13
16
2           top[] size m = 2
10          top == [10, 15]
15
```

Sample Output

```
9
```

Explanation

The first time requests are served, at *time* = 10, the 5 requests received at [3, 4, 5, 6, 6] are served and deleted from the queue. Now *timestamp'* = [1, 2, 2, 13, 16].

The next time the server has priority, at *time* = 15, it serves the 4 requests received at [1, 2, 2, 13] respectively and deletes them.

A total of 9 requests were served.

Since the server did not have priority after *time* = 15, the request received at *time* = 16 was not served.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
7           timestamp[] size n = 7
2           timestamp = [2, 2, 4, 8, 11, 28, 30]
2
4
8
11
28
30
4           top[] size m = 4
0           top = [0, 5, 5, 15]
5
5
15
```

Sample Output

```
5
```

Explanation

At the first timestamp, the server has priority at *time* = 0. Since there are no pending requests, none are served.

The second time, *time* = 5, the 3 requests received at [2, 2, 4] are served.

The third time, again at *time* = 5, there are no pending requests, so none are served.

The fourth time, *time* = 15, the 2 requests received at [8, 11] are served.

A total of 5 requests were served.

▼ Sample Case 2

Sample Input For Custom Testing

```
17
0
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
4
6
6
6
6
```

Sample Output

```
17
```

Explanation

The server has priority at *time* = 6; it serves the latest 5 requests received at *time* = 1.

The second time, *time* = 6, the next 5 requests received at *time* = 1 are served.

The third time, *time* = 6, the next 5 requests received at *time* = 1 are served.

The fourth time, again at *time* = 6, the 2 remaining requests received at *time* = 0 are served.

A total of 17 requests were served.

Question - 50
Oscillating String

Sam and Alex are competitive coders who enjoy creating string challenges for each other. In one such challenge, Sam asks Alex to write a function that sorts a string according to specific rules. Here are the rules for forming the sorted string:

1. The sorted string, *ss*, begins with the smallest character in the original string, *s*.
2. Select the smallest character from the remaining string that is larger than the last appended character, and add it to *ss*.
3. Repeat step 2 until no more characters can be selected.
4. Choose the largest character from the remaining string that is smaller than the last appended character, and add it to *ss*.
5. Repeat step 4 until no more characters can be selected.
6. If all remaining characters are equal and no character was chosen in steps 2 through 5, pick any one of the equal characters and add it to *ss*.
7. Repeat steps 2 through 6 until all characters in the string have been processed.

Example
s = ababyz

The following table shows the method:

```
s      choose ss
ababyz a      a       <- step 1
babyz  b      ab      <- step 2
abyz   y      aby     <- repeat step 2
abz    z      abyz    <- repeat step 2, reached the end
ab     b      abyzb   <- step 4
a      a      abyzba  <- repeat step 4, reached the end
```

The final sorted string is *abyzba*.

Function Description
Complete the function *formString* in the editor with the following parameter:

string s: a string of characters

Returns
string: the characters sorted per the rules

Constraints
- $0 < |s| \leq 10^5$
- All letters in *s* are in the range *ascii[a-z]*.

▼ Input Format For Custom Testing

The only line contains a string, *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
babcbb   →  s = "babcbb"
```

Sample Output

```
abcbbb
```

Explanation

```
s       choose ss
babcbb  a      a        <- step 1
bbcbb   b      ab       <- step 2
bcbb    c      abc      <- step 2, reached the end
bbb     b      abcb     <- step 4, reached the end
bb      b      abcbb    <- step 6, no lesser or greater character so append b
b       b      abcbbb   <- step 6, no lesser or greater character so append b
```

The final sorted string is abcbbb.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-------    -----------
zafb    → s = "zafb"
```

Sample Output

```
abfz
```

Explanation

```
s       choose ss
zafb    a      a        <- step 1
zfb     b      ab       <- step 2
zf      f      abf      <- step 2
z       z      abfz     <- step 2, reached the end
```

There are no characters left after repeating step 2, so step 4 is not necessary. The final sorted string is abfz.

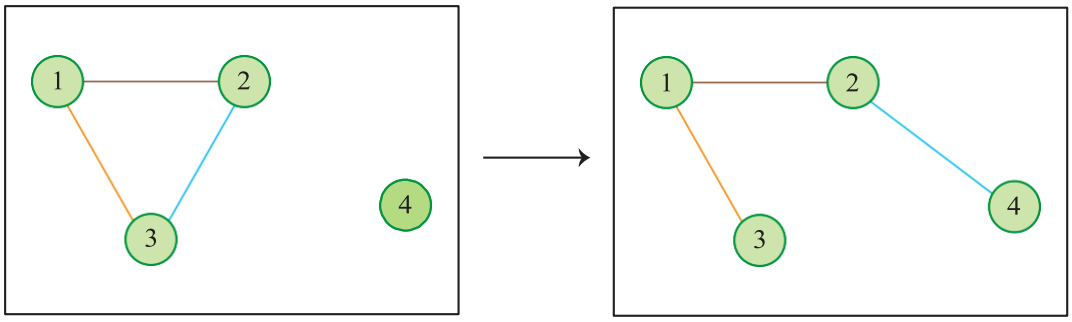
Question - 51
Connecting Computers

A set of computers may be connected by ethernet cables, with each cable connecting exactly two distinct computers. Computers are considered connected if they have either a direct or indirect connection to each other.

Initially, some groups of computers are connected to each other, but others may be disconnected. You can perform one type of operation: remove a cable between any two computers and use it to connect any other pair of computers.

Determine the minimum number of such operations required to connect all computers. If it is not possible to connect all computers, return -1.

Example
comp_nodes = 4
comp_edges = 3
c_from = [1, 1, 3]
c_to = [2, 3, 2]



Initially, there are connections between computers (1,2), (1,3), and (2,3), but computer 4 is disconnected. By removing the cable between computers 2 and 3 and placing it between computers 2 and 4, all computers become connected.

The minimum number of operations required is 1.

Function Description

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

int comp_nodes: the number of computers
int comp_from: one end of each connection
int comp_to: the other end of each connection

Returns

int: the minimum operations to connect the network or -1 if it is not possible

Constraints

- $2 \leq comp\_nodes \leq 2 * 10^5$
- $1 \leq comp\_edges \leq 3 * 10^5$
- $1 \leq comp\_from[i] \leq comp\_nodes$
- $1 \leq comp\_to[i] \leq comp\_nodes$

▼ Input Format For Custom Testing

The first line contains two space-separated integers, *comp_nodes* and *comp_edges*.

Each line *i* of the *comp_edges* subsequent lines (where $0 \leq i < comp\_edges$) contains 2 space-separated integers, *comp_from[i]* and *comp_to[i]*.

▼ Sample Case 0

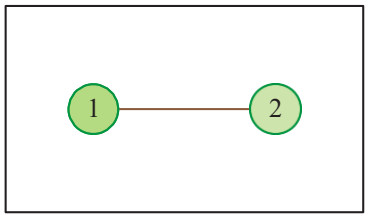
Sample Input For Custom Testing

```
STDIN          Function
-----          --------
2 1      →     comp_nodes = 2, comp_edges = 1
1 2      →     comp_from[ ] = [ 1 ], comp_to[ ] = [ 2 ]
```

Sample Output

```
0
```

Explanation



All the computers are connected.

▼ Sample Case 1

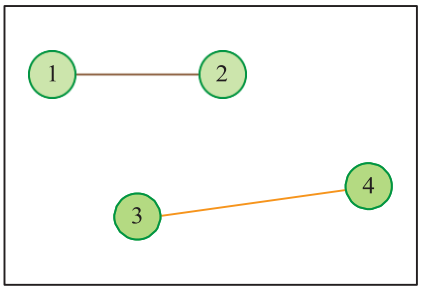
Sample Input For Custom Testing

```
STDIN          Function
-----          --------
4 2      →     comp_nodes = 4, comp_edges = 2
1 2      →     comp_from[ ] = [ 1, 2 ], comp_to[ ] = [ 3 , 4 ]
3 4
```

Sample Output

```
-1
```

Explanation



It is not possible to connect all computers.

Question - 52
Longest K-Interspace Substring

A string is a "*k*-interspace" string if the absolute difference in ASCII values between every pair of adjacent characters is at most *k*.
Given a string *word* and an integer *k*, find the longest *k*-interspace substring within *word*. If there are multiple substrings of the longest length, return the one that occurs first.

Examples

word = "wedding"
k = 0
The first occurring longest 0-interspace substring is "dd".

word = "ababbacaabbbb"
k = 1
There are two 1-interspace substrings of length 6: "ababba" and "aabbbb". The one that occurs first is "ababba".

Function Description

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

string word: the string to analyze

int k: maximum difference in character values

Returns

string: the first occurring longest *k*-interspace substring of the word

Constraints

- $1 \leq |word| \leq 10^6$
- $0 \leq k \leq 25$
- *word* contains only lowercase English letters.

▼ Input Format For Custom Testing

The first line contains a string, *word*.
The second line contains an integer, *k*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          Function
-----          --------
hackerrank  → word = "hackerrank"
0           → k = 0
```

Sample Output

```
rr
```

Explanation

The first occurring longest 0-interspace substring is "rr".

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          Function
-----          --------
ababbaca    → word = "ababbaca"
1           → k = 1
```

Sample Output

```
ababba
```

Explanation

The first occurring longest 1-interspace substring is "ababba".

Question - 53
Divided Matrix

Given a square matrix of integers, divide it along its two diagonals into four distinct regions. Calculate the sum of elements in each region and return these sums as an array.

The four regions are:

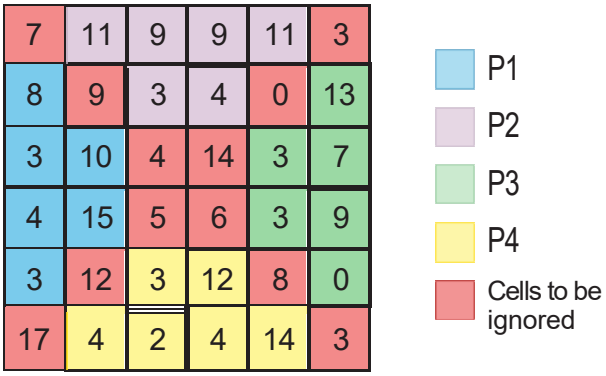
- P1: Elements to the left of both diagonals
- P2: Elements above both diagonals
- P3: Elements to the right of both diagonals
- P4: Elements below both diagonals

Elements on the diagonals themselves are excluded from all sums.

Example

arr = [[7, 11, 9, 9, 11, 3], [8, 9, 3, 4, 0, 13], [3, 10, 4, 14, 3, 7], [4, 15, 5, 6, 3, 9], [3, 12, 3, 12, 8, 0], [17, 4, 2, 4, 14, 3]]

The matrix is represented below.



The sums of the four regions are:

- P1 (left) = 8 + 3 + 4 + 3 + 10 + 15 = 43
- P2 (top) = 11 + 9 + 9 + 11 + 3 + 4 = 47
- P3 (right) = 3 + 3 + 13 + 7 + 9 + 0 = 35
- P4 (bottom) = 3 + 12 + 4 + 2 + 4 + 14 = 39

The return value is [43, 47, 35, 39].

Function Description

Complete the function *divided* in the editor below. The function must return an array that contains *4* integers.

divided has the following parameter(s):

int arr[n][n]: a square 2D array of integers

Returns

int[4]: the values of *P1, P2, P3, P4* in that order

Constraints

- $3 \le n \le 1000$
- $0 \le arr[i][j] \le 1000$
- The number of rows = the number of columns.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of rows in the 2D array *arr*.

The second line repeats the integer *n* and denotes the number of columns in *arr.*

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

▼ Sample Case 0

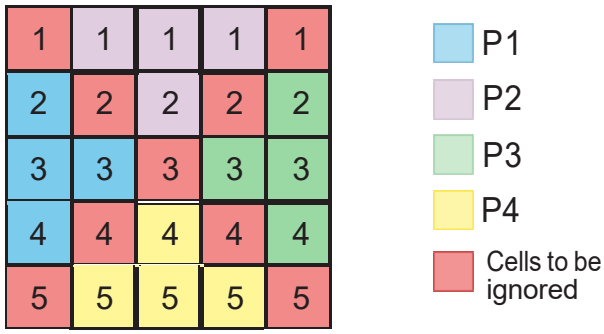
Sample Input For Custom Testing

```
STDIN          Function
-----          --------
5          →   arr[] size n = 5 (rows)
5          →   arr[i][] size n = 5 (columns)
1 1 1 1 1 →    arr = [[1, 1, 1, 1, 1], [2, 2, 2, 2, 2], [3, 3, 3, 3, 3], [4, 4, 4, 4, 4], [5, 5, 5, 5, 5]]
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Sample Output

```
12
5
12
19
```

Explanation



The sum of *P1* = 2 + 3 + 4 + 3 = 12
The sum of *P2* = 1 + 1 + 1 + 2 = 5
The sum of *P3* = 2 + 3 + 4 + 3 = 12
The sum of *P4* = 5 + 5 + 5 + 4 = 19
The return value is [12, 5, 12, 19].

▼ Sample Case 1

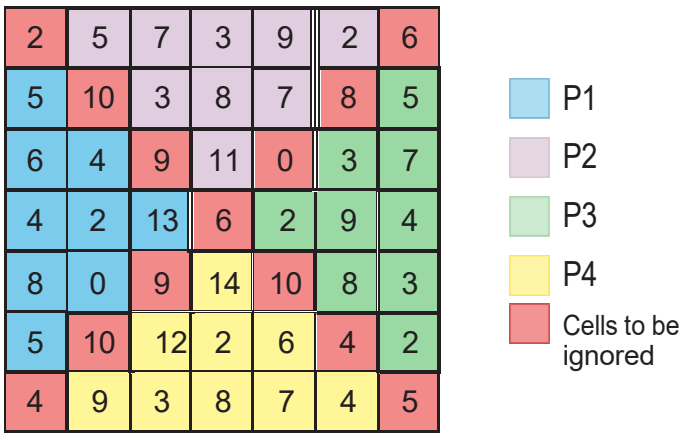
Sample Input For Custom Testing

```
STDIN                Function
-----                --------
7                 → arr[] size n = 7
7                 → arr[i][] size n = 7
2 5 7 3 9 2 6     → arr = [[2, 5, 7, 3, 9, 2, 6], [5, 10, 3, 8, 7, 8, 5], [6, 4, 9, 11, 0, 3, 7],\
5 10 3 8 7 8 5             [4, 2, 13, 6, 2, 9, 4], [8, 0, 9, 14, 10, 8, 3], [5, 10, 12, 2, 6, 4, 2], [4, 9, 3, 8,
7, 4, 5]]
6 4 9 11 0 3 7
4 2 13 6 2 9 4
8 0 9 14 10 8 3
5 10 12 2 6 4 2
4 9 3 8 7 4 5
```

Sample Output

```
47
55
43
65
```

Explanation



The sum of *P1* = 5 + 6 + 4 + 8 + 5 + 4 + 2 + 0 + 13 = 47
The sum of *P2* = 5 + 7 + 3 + 9 + 2 + 3 + 8 + 7 + 11 = 55

The sum of *P3* = 2 + 3 + 9 + 8 + 5 + 7 + 4 + 3 + 2 = 43
The sum of *P4* = 14 + 12 + 2 + 6 + 9 + 3 + 8 + 7 + 4 = 65
The return value is [47, 55, 43, 65]

▼ Sample Case 2

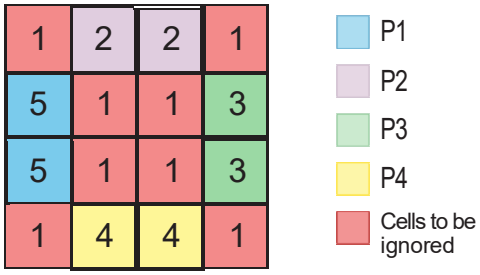
Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4       →   arr[] size n = 4
4       →   arr[i][] size n = 4
1 2 2 1 →   arr = [[1, 2, 2, 1], [5, 1, 1, 3], [5, 1, 1, 3], [1, 4, 4, 1]]
5 1 1 3
5 1 1 3
1 4 4 1
```

Sample Output

```
10
4
6
8
```

Explanation



In this scenario, there is an even number of rows and columns. The diagonal values are all 1.

The sum of *P1* = 5 + 5 = 10
The sum of *P2* = 2 + 2 = 4
The sum of *P3* = 3 + 3 = 6
The sum of *P4* = 4 + 4 = 8
The return value is [10, 4, 6, 8]

Question - 54
Equalizing Power

Researchers are investigating the logistics of robot swarms. In a specific experiment, they need to ensure that each robot starts with the same power level before the swarm is deployed. They have a charging device that can increase or decrease the power of one robot by 1 unit per minute. Determine the minimum number of minutes required to equalize the power levels of a swarm of *n* robots.

Example
n = 6
power = [2, 5, 3, 6, 7, 1]

There are *n* = 6 robots in the swarm. If all of them are adjusted to 4 units, it takes (4 - 2) + (5 - 4) + (4 - 3) + (6 - 4) + (7 - 4) + (4 - 1) = 2 + 1 + 1 + 2 + 3 + 3 = 12 minutes. This is one way to change them in the minimum time. The return value is 12.

Function Description

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

int power[n]: the power levels of the robots

Returns

int: the minimum time required to equalize the robots' power

Constraints

- $1 \le n \le 10^5$
- $0 \le power[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in the array *power*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *power[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
5       →   power[] size n = 5
1       →   power = [1, 2, 3, 4, 5]
2
3
4
5
```

Sample Output

```
6
```

Explanation

It takes 6 minutes to equalize their power at 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
10      →   power[] size n = 10
10      →   power = [10, 20, 30, 40, 50, 51, 52, 53, 54, 55]
20
30
40
50
51
52
53
54
55
```

Sample Output

```
115
```

Explanation

One way to minimize time is to adjust them to 50.

Question - 55

Merge Sort Counts

Implement a modified merge sort algorithm that counts specific inversions during the sorting process:

1. The algorithm takes an array of unique integers as input and returns the sorted array.

2. For an array with n elements:
   - If the array has fewer than 2 elements, it is returned as is.
   - Otherwise, it is split into left and right arrays.
   - The left array contains the first half of elements (including the middle element if n is odd).
   - The right array contains the remaining elements.
3. The algorithm recursively sorts both arrays.
4. During the merging process:
   - A count is maintained for each element in the input array (initially 0).
   - Whenever an integer k from the right array is merged before any element from the left array, 1 is added to the count.

Return the maximum count value after the merge sort algorithm completes.

Example

arr = [2, 3, 1]

Starting with array [2, 3, 1] and all counters initialized to 0:

1. Initial division
   - Divide into left = [2, 3] and right = [1]
2. Sort the left subarray [2, 3]
   - Further divide into [2] and [3]
   - Since both are single-element arrays, they are already sorted
   - Merge [2] and [3] into [2, 3]
   - During this merge, 3 comes after 2 (following the correct order), so no counter increment
3. Sort the right subarray [1]
   - Since [1] has only one element, it is already sorted
4. Final merge of [2, 3] and [1]
   - When merging, we compare elements from both arrays and take the smaller one first.
   - Compare 2 (from left) with 1 (from right): 1 is smaller
   - Since an element from the right array (1) is placed before an element from the left array (2), increment the counter to 1.
   - Complete the merge by adding the remaining elements: [1, 2, 3]

The final sorted array is [1, 2, 3]. The maximum counter value is 1.

Function Description

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

int arr[n]: array of distinct elements

Returns

int: the maximum count value after the merge sort completes

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$
- All elements of *arr* are distinct.

▼ Input Format Format for Custom Testing

In the first line, there is a single integer n.

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

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
3       →   arr[] size n = 3
2       →   arr = [ 2, 1, 3]
1
3
```

Sample Output

```
1
```

Explanation

Starting with array [2, 1, 3] and all counters initialized to 0:

1. Initial division
   - Divide into left = [2, 1] and right = [3]
2. Sort the left subarray [2, 1]
   - Further divide into [2] and [1]
   - Since both are single-element arrays, they are already sorted.
   - Compare 2 (from left) with 1 (from right): 1 is smaller.
   - Since an element from the right array (1) is placed before an element from the left array (2), increment the counter to 1.
3. Sort the right subarray [3]
   - Since [3] has only one element, it is already sorted.
4. Final merge of [1, 2] and [3]
   - When merging, we compare elements from both arrays and take the smaller one first.
   - Compare 1 (from left) with 3 (from right): 1 is smaller.
   - During this merge, 3 comes after 1 (following the correct order), so there is no counter increment.
   - Complete the merge by adding the remaining elements: [1, 2, 3]. The final sorted array is [1, 2, 3]. The maximum counter value is 1.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
6      →    arr[] size n = 6
1      →    arr = [1, 2, 3, 4, 5, 10]
2
3
4
5
10
```

Sample Output

```
0
```

Explanation

Starting with array [1, 2, 3, 4, 5, 10] and all counters initialized to 0:

1. Initial division
   - Divide into left = [1, 2, 3] and right = [4, 5, 10]
2. Sort the left subarray [1, 2, 3]
   - Further divide into [1, 2] and [3]
   - For [1, 2], divided into [1] and [2], both are already sorted
   - Merge [1] and [2] into [1, 2] (no counter increments)
   - [3] is already sorted
   - Merge [1, 2] and [3] into [1, 2, 3] (no counter increments)
3. Sort the right subarray [4, 5, 10]
   - Similar process yields [4, 5, 10] (no counter increments)
4. Final merge of [1, 2, 3] and [4, 5, 10]
   - When merging, we compare elements from both arrays and take the smaller one first.
   - All elements from the left array are smaller than the right array elements
   - Complete the merge in order: [1, 2, 3, 4, 5, 10]

- No elements from the right array are placed before elements from the left array, so there are no counter increments. The final sorted array is [1, 2, 3, 4, 5, 10]. The maximum counter value is 0.

Question - 56
Median of 3 Pivots

In this variation of the quicksort algorithm, a global array pivots is initialized as empty. The algorithm:

1. If the array has fewer than 3 elements, sorts it using a non-recursive method
2. Otherwise, selects a pivot *p* as the median of the first, middle, and last elements of the array
   - For an array with *m* elements, selects the median of elements at indices 0, ⌊*m*/2⌋, and *m-1*. Note that ⌊*m*/2⌋ means the floor or integer part of *m*/2, e.g., floor(5/2) = 2 and floor(4/2) = 2.
   - Appends *p* to the pivots array
3. Partitions the array into two subarrays:
   - *left*: elements less than *p* in their original relative order
   - *right*: elements larger than *p* in their original relative order
4. Recursively calls itself on the *left* array
5. Recursively calls itself on the *right* array

Determine the sequence of pivot elements chosen by this algorithm.

Example
arr = [8, 4, 3, 1]

- The array has 4 elements, so the algorithm selects *p* as the median of 8, 3, and 1 (first, middle, last).
- The median is 3, which is appended to the *pivots* array.
- Recursive calls are made with arrays *left* = [1] and *right* = [8, 4].
- No new pivots are produced from these calls.
- The final *pivots* array is [3].

Function Description
Complete the function *quicksortMedianOf3Pivots* in the editor with the following parameter(s):

int arr[n]: distinct integers

Returns
int[]: the pivots produced by the algorithm in the order generated

Constraints
- $3 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$
- The elements of *arr* are chosen uniformly at random without replacement from the range $[1, 10^9]$.

▼ Input Format Format for Custom Testing
In the first line, there is a single integer *n,* the size of *arr[]*.

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

▼ Sample Case 0
Sample Input

```
STDIN       Function
-----       --------
3           arr[] size n = 3
3           arr = [3, 1, 2]
1
2
```

Sample Output

```
2
```

Explanation

Take the median of the three values, i.e., 2. Add 2 to the pivots array, *left* = [1] and *right* = [3]. Neither of these requires a pivot, so return [2].

▼ Sample Case 1

Sample Input

```
6
5
2
3
16
1
4
```

Sample Output

```
5
2
```

Explanation

There are 6 elements, so take the median of values at indices 0, 3, and 5, i.e., 5, 16, and 4. The median is 5, which is added to the pivots array.
Now *left* = [2, 3, 1, 4] and *right* = [16].
With *left*, take the median of values at indices 0, 2, and 3, i.e., 2, 1, and 4. The median is 2, which is added to the pivots array.
Now *left* = [1] and *right* = [3, 4].
No more recursive pivots are required.

Question - 57
Fill Missing Brackets

Determine the number of ways a string containing the characters '(', ')', '[', ']', and '?' can be divided into two non-empty substrings such that each substring can be rearranged to form a balanced string. The '?' characters can be replaced with any character needed to achieve balance. The two substrings together must cover the entire original string.

A balanced string has properly matched pairs of brackets without overlaps. For example, "([])" is balanced, but "([)]" is not.

Note: A substring is defined as a contiguous sequence of characters within the original string.

Example

s = '[(?][??['

The string *s* can be split into two balanced substrings as follows:

1. *s1 = [(?]* and *s2 = [??[*:
   - Replace the ? in *s1* with) to have a balanced string [()].
   - Replace the question marks in *s2* with] and rearrange to make [[]] or [][].
2. *s1 = [(?][?* and *s2 = ?]*:
   - Replace the ?s in *s1* to get [()][].
   - Replace the ? in *s2* with [to get [].

Constraints

- *s* contains *(,), [,]* and *?* only
- $4 \leq length\ of\ s \leq 10^5$

▼ Input Format For Custom Testing

The only line contains the string *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
(?][   → s = '(?]['
```

Sample Output

```
1
```

Explanation

The only possible split is:
s1 = (? Replace the question mark in *s1* to get ().
s2 =][Rearrange *s2* to get [].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
(((?   → s = '(((?'
```

Sample Output

```
0
```

Explanation
s = ((((?

There are 3 splits into two non-empty substrings:

1. (and ((?
2. ((and (?
3. (((and ?

None has two balanced substrings.

Question - 58
Counting Analogous Arrays

A secret agent has important information stored in an array of integers that must be kept confidential. However, the following details about the array are known:

An array is considered analogous to the secret array if all these conditions are satisfied:

- The array's length matches the secret array's length.
- Each integer in the array falls within the inclusive range [*lowerBound, upperBound*].
- The difference between each pair of consecutive integers in the array matches the difference between the corresponding pair in the secret array. In other words, if the secret array is *[s[0], s[1],...s[n-1]]* and the analogous array is *[a[0], a[1],...a[n-1]]*, then *(a[i-1] - a[i])* must equal *(s[i-1] - s[i])* for each *i* from 1 to *n-1*.

Given the integers *lowerBound* and *upperBound*, inclusive, and the array of differences between each pair of consecutive integers in the secret array, determine the number of arrays that are analogous to the secret array. If no such array exists, return 0.

For example:
consecutiveDifference = [-2, -1, -2, 5]
lowerBound=3
upperBound=10

The logic to create an analogous array starting from the lower bound is:

- Start with a value of 3.
- Subtract *consecutiveDistances[0]*, 3 - (-2) = 5
- Subtract *consecutiveDistances[1]*, 5 - (-1) = 6
- Subtract *consecutiveDistances[2]*, 6 - (-2) = 8
- Subtract *consecutiveDistances[3]*, 8 - 5 = 3

Note that none of the values is out of bounds. All possible analogous arrays are:
[3, 5, 6, 8, 3]
[4, 6, 7, 9, 4]
[5, 7, 8, 10, 5]

The answer is *3*.

Function Description

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

int *consecutiveDifference[n]*: the differences between each pair of consecutive integers in the secret array
int *lowerBound*: an integer
int *upperBound*: an integer

Returns

int: the number of arrays that are analogous to the secret array

Constraints

- $-10^9 \leq lowerBound \leq upperBound \leq 10^9$
- $1 \leq n \leq 10^5$
- $-2*10^9 \leq consecutiveDifference[i] \leq 2*10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *consecutiveDifference*.
Each line *i* of the *n* subsequent lines contains an integer, *consecutiveDifference[i], where 0 ≤ i < n*.
The next line contains an integer, *lowerBound*.
The last line contains an integer, *upperBound*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       ------------
3      →    consecutiveDifferences[] size n = 3
-1     →    consecutiveDifferences = [-1, -3, 2]
-3
2
2      →    lowerBound = 2
8      →    upperBound = 8
```

Sample Output

```
3
```

Explanation

There are three analogous arrays:

[3, 4, 7, 5]
[2, 3, 6, 4]
[4, 5, 8, 6]

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
2     →     consecutiveDifferences[]  size n = 2
1     →     consecutiveDifferences =  [1, 2]
2
3     →     lowerBound = 3
4     →     upperBound = 4
```

Sample Output

```
0
```

Explanation

There is no analogous array.

Question - 59
Missed Chapters

A Chemistry teacher organizes lectures where each chapter is taught in sequence, one per day, repeating from the beginning after reaching the end of the book. Formally, with *numChapters* in the book, on the i^{th} day (zero-indexed), the lecture covers chapter *i % numChapters* where % is the modulo operator.

A student will miss classes from *firstDay* to *lastDay,* inclusive. Calculate the number of different chapters the student will miss lectures on during this absence.

For example, there are *numChapters = 4* chapters in the book. The student is out of class beginning on day *firstDay = 3* and ending on day *lastDay = 5.* The series of lectures are on chapters *class = [0,1, 2, 3, 0, 1, 2, 3,...]* starting from day *0.* For *class[3]* through *class[5],* lectures are given on chapters *3, 0,* and *1.* The student will miss lectures on *3* chapters.

Function Description

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

missedLectures has the following parameters:

int numChapters: the number of chapters
int firstDay: the first day the student is out
int lastDay: the last day the student is out

Returns

int: the number of chapters missed

Constraints

- $1 \le numChapters \le 10^9$
- $1 \le firstDay \le lastDay \le 10^9$

▼ Input Format For Custom Testing

There are three lines of input, each with a single integer *numChapters, firstDay,* and *lastDay*, respectively.

▼ Sample Case 0

Sample Input For Custom Testing

```
5
5
6
```

Sample Output

```
2
```

Explanation

Chapters are taught in the order *class = [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0,...]*. The student will miss chapters *0* and *1* on days *5* and *6*, respectively.

▼ Sample Case 1

Sample Input For Custom Testing

```
5
13
98
```

Sample Output

```
5
```

Explanation

Chapters are taught in the order *class = [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0,...]*. The student will miss each of the *5* chapters at least once between *class[13]* and *class[98]*.

Question - 60

Dam Design

Your company is planning to build a dam across a stream to form a small lake. To minimize material costs, the dam will consist of one or more concrete walls with mud packed in between. Determine the maximum height of the mud segments in the dam given the following constraints:

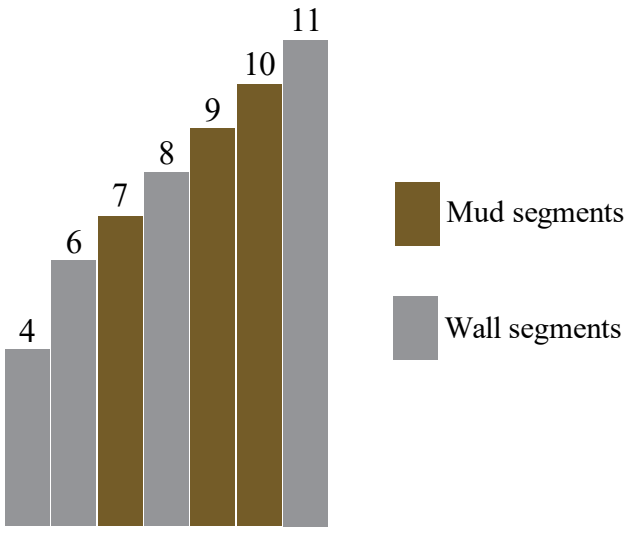
- One unit width of the gap between walls will contain one segment of packed mud.
- The height of the mud in a segment cannot exceed 1 unit more than an adjacent wall or mud segment.

Given the positions and heights of a number of walls, determine the maximum height of any mud segment that can be constructed. If no mud segment can be built, return 0.

Example

wallPositions = [1, 2, 4, 7]

wallHeights = [4, 6, 8, 11]



- There is no space between the first two walls.
- Between positions *2* and *4*, there is one unit open for mud. The heights of the surrounding walls are *6* and *8*, so the maximum height of mud is *6 + 1 = 7*.
- Between positions *4* and *7* there are two units. The heights of the surrounding walls are *8* and *11*.
  - The maximum height mud segment next to the wall of height *8* is *9*.
  - The maximum height of mud next to a mud segment of height *9* is *10*.
- Overall, mud segment heights are *7*, *9*, and *10*, and the maximum height is *10*.

Function Description

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

int *wallPositions[n]:* an array of integers

int *wallHeights[n]:* an array of integers

Returns

int: the maximum height mud segment that can be built

Constraints

- $1 < n \leq 10^5$
- $1 \leq wallPositions[i], wallHeights[i] \leq 10^9$ *(where* $0 \leq i < n$*)*

▼ Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, *wallPositions[i]*.

The next line contains the integer, *n*, the number of elements in *wallHeights*.

Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, *wallHeights[i]*.

▼ Sample Case 0

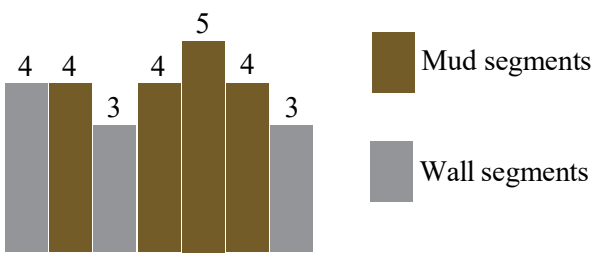
Sample Input For Custom Testing

```
STDIN       Function
-----       -----------
3     →     wallPositions[] size n = 3
1     →     wallPositions = [1, 3, 7]
3
7
3     →     wallHeights[] size n = 3
4     →     wallHeights = [4, 3, 3]
3
3
```

Sample Output

```
5
```

Explanation



There can be a segment of height *4* at position *2* supported by walls of heights *4* and *3*. Between positions *3* and *7*, there can be a segment of height *4* at positions *4* and *6*. Between them, a segment can be built of height *5* at position *5*.

▼ Sample Case 1

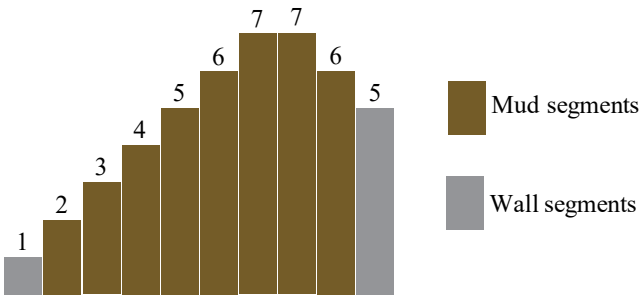
Sample Input For Custom Testing

```
STDIN       Function
-----       --------
2       →   wallPositions[] size n = 2
1       →   wallPositions = [1, 10]
10
2       →   wallHeights[] size n = 2
1       →   wallHeights = [1, 5]
5
```

Sample Output

```
7
```

Explanation



The heights of the mud segments from positions *2* through *9* are *[2, 3, 4, 5, 6, 7, 7, 6]*.

Question - 61
Math Homework

Students have been given a series of math problems, each with a specific point value. Given a sorted array of these point values, determine the minimum number of problems a student needs to solve based on the following rules:

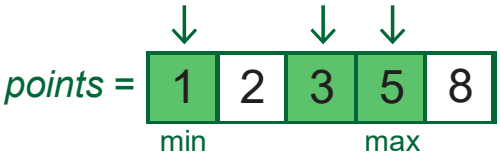
1. The student must always solve the first problem (at index *i* = 0).
2. After solving the ith problem, the student can either solve the next problem (*i+1*) or skip ahead to the (*i+2*) problem.
3. The student must continue solving problems until the difference between the maximum and minimum points of the solved problems meets or exceeds a given threshold.
4. If the threshold cannot be met or exceeded, the student must solve all the problems.

Return the minimum number of problems the student needs to solve.

Example

threshold = 4

points = [1, 2, 3, 5, 8]



If a student solves *points[0]* = 1, *points[2]* = 3, and *points[3]* = 5, then the difference between the minimum and the maximum points solved is 5 - 1 = 4. This meets the threshold, so the student must solve at least 3 problems, so return 3.

If instead the threshold is 7, solve problems 0, 2, and 4, where *points[4] - points[0]* = *8 - 1* = *7*. Again, the student must solve 3 problems.

There is no way to meet a threshold greater than 7. In that case, all problems need to be solved, and the return value is 5.

Function Description

Complete the function *minNum* in the editorwith the following parameters:

int threshold: the minimum difference required

int points[n]: a sorted array of integers

Returns

int: the minimum number of problems that must be solved

Constraints

- $1 \le n \le 100$
- $1 \le points[i] \le 1000$
- $1 \le threshold \le 1000$

▼ Input Format For Custom Testing

The first line contains an integer, *threshold*.

Next line contains an integer, *n*, the size of the *points* array.

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

▼ Sample Case 0

Sample Input For Custom Testing

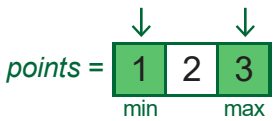
```
STDIN       Function
-----       --------
2       →   threshold = 2
3       →   points[] size n = 3
1       →   points = [1, 2, 3]
2
3
```

Sample Output

```
2
```

Explanation

- The student chooses *points[0]* = 1 and *points[2]* = 3.
- The difference between the minimum and the maximum points is 3 - 1 = 2.
- This meets the threshold, and the return value is 2.



▼ Sample Case 1

Sample Input For Custom Testing

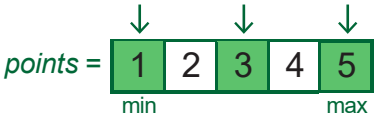
```
STDIN       Function
-----       --------
4       →   threshold = 4
5       →   points[] size n = 5
1       →   points = [1, 2, 3, 4, 5]
2
3
4
5
```

Sample Output

```
3
```

Explanation

- The student should choose *points[0]* = 1, *points[2]* = 3 and *points[4]* = 5.
- The difference between the minimum and the maximum points solved is 5 - 1 = 4.
- This meets the threshold and the return value is 3.



Question - 62
Better Compression

You are given a string *S* that represents a compressed string format. Each character is followed by an integer indicating its frequency. However, the string may not be properly compressed, i.e., the same character might appear multiple times in different places.

Your task is to create a properly compressed string where:

- Each character appears exactly once.
- Characters are arranged in alphabetical order.
- The frequency for each character is the sum of all its occurrences in the original string,

Example:

S = "a3c9b2c1"

The string is not properly compressed because 'c' appears twice with frequencies 9 and 1. The properly compressed version is "a3b2c10" where:

- 'a' appears once with frequency 3
- 'b' appears once with frequency 2
- 'c' appears once with combined frequency 10 (9+1)
- All characters are in alphabetical order (a, b, c)

Function Description

Complete the function *betterCompression* in the editor with the following parameter:

string *S*: a compressed string

Returns

string: the properly compressed string

Constraints

- 1 ≤ size of *S* ≤ 100000
- All characters in *S* are lowercase English letters, 'a'-'z'.
- 1 ≤ frequency of each character in *S* ≤ 1000

▼ Input Format For Custom Testing

The only line contains a string, *S*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     --------
a12b56c1 → S = 'a12b56c1'
```

Sample Output

```
a12b56c1
```

Explanation
Nothing is changed because each character occurred only once, and they are already sorted in ascending order.

▼ Sample Case 1

Sample Input

```
STDIN         Function
-------       --------
a12c56a1b5 → S = 'a12c56a1b5'
```

Sample Output

```
a13b5c56
```

Explanation
'a' occurs twice: 12 times for the first occurrence and 1 time in the second occurrence for a total 13. Sort 'b' and 'c' in order in the final compression.

Question - 63
Shortest Substring Containing Characters

Given a string made up of lowercase letters from 'a' to 'z', determine the length of the shortest substring that includes at least one of each distinct letter present in the string.

Example
givenString = 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 that contains all of the letters is *4* characters long. Return *4* as the answer.

Function Description

Complete the function *shortestSubstring* in the editor with the following parameter(s):
string *givenString:* the string

Returns
int: the length of the shortest substring that contains at least one of each character in *givenString*

Constraints
- $1 \le$ *size of givenString* $\le 10^5$
- each *givenString[i]* is in the set *ascii[a-z]*

▼ Input Format For Custom Testing

The first line contains a string, *givenString*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
bab    →    givenString = 'bab'
```

Sample Output

```
2
```

Explanation

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

▼ Sample Case 1

Sample Input

```
STDIN                       Function
-----                       --------
asdfkjeghfalawefhaef →      givenString = 'asdfkjeghfalawefhaef'
```

Sample Output

```
13
```

Explanation

The *11* distinct characters in *givenString* are *[a, d, e, f, g, h, j, k, l, s, w]*. The shortest substring with all of the characters is 13 characters long: *sdfkjeghfalaw*.

Question - 64

Maximize the Value

Rearrange an array of integers to maximize the calculated value *U*. The value of *U* for an array with *n* elements is calculated as follows:

If *n* is odd: $U = arr[1] \times arr[2] \times (1 \div arr[3]) \times arr[4] \times ... \times arr[n-1] \times (1 \div arr[n])$
If *n* is even: $U = arr[1] \times arr[2] \times (1 \div arr[3]) \times arr[4] \times ... \times (1 \div arr[n-1]) \times arr[n]$

The sequence of operations alternates between multiplication and division, starting with multiplication. The length of the array determines whether the calculation ends with multiplication or division.

Among the possible arrangements that maximize *U*, choose the array with the smallest numerical order.

Example

arr = [21, 34, 5, 7, 9]

To maximize *U* and minimize the order, arrange the array as [9, 21, 5, 34, 7] so $U = 9 \times 21 \times (1 \div 5) \times 34 \times (1 \div 7) = 183.6$. The same *U* can be achieved using several other orders, e.g., [21, 9, 7, 34, 5] = $21 \times 9 \times (1 \div 7) \times 34 \times (1 \div 5) = 183.6$, but they are not in the minimum order.

Function Description

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

int arr[n]: an array of integers

Returns

int[n]: the elements of *arr* rearranged as described

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[i] \le 10^9$

▼ 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* $1 \le i \le n$) contains an integer, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
4     →     arr[] size n = 4
1     →     arr = [1, 2, 3, 4]
2
3
4
```

Sample Output

```
2
3
1
4
```

Explanation

U = 2×3×(1÷1)×4 = 24. All other arrangements where *U* = 24 are numerically higher than this array, e.g., [2, 3, 1, 4] < [3, 4, 1, 2].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
2     →     arr[] size n = 2
4     →     arr = [4, 5]
5
```

Sample Output

```
4
5
```

Explanation

U is always 4×5 = 20, and [4, 5] < [5, 4].

Question - 65
Air Invaders

You are tasked with developing a defense strategy for a video game involving enemy aircraft. Each aircraft has a specific starting height above ground and descends at a constant rate. Your character can shoot down one aircraft per second. The game ends when any aircraft successfully lands on the ground.

Your goal is to determine the maximum number of aircraft that can be prevented from landing.

For each aircraft, you know:

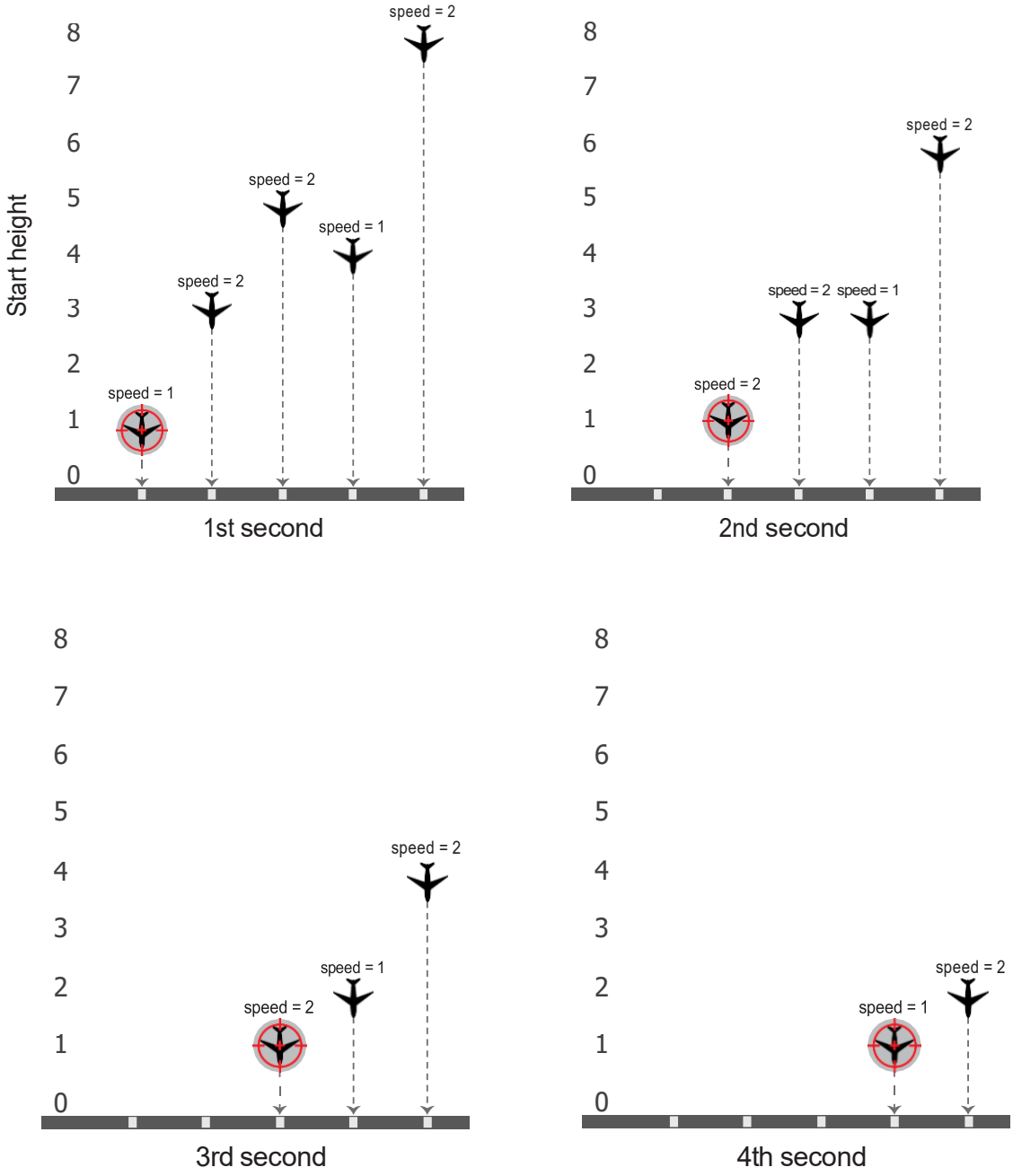
- Its initial height above ground
- Its rate of descent (units per second)

Example

n = 5 aircraft

startHeight = [1, 3, 5, 4, 8]

descentRate = [1, 2, 2, 1, 2]



- Aircraft 1: Starts at height 1, falls at 1 unit per second. It must be shot during the first second.
- Aircraft 2: Starts at height 3, falls at 2 units per second. It must be shot during the second second (before landing at t= 1.5).
- Aircraft 3: Starts at height 5, falls at 2 units per second. It must be shot during the third second (before landing at t = 2.5).
- Aircraft 4 or 5: One of these must be shot during the fourth second (before either lands at t = 4.0).
- The remaining aircraft cannot be shot down.

The maximum number of aircraft that can be prevented from landing is 4.

Function Description

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

int startHeight[n]: the starting heights of each aircraft

int descentRate[n]: the units per second that each aircraft is descending

Returns

int: the maximum number of planes the character can prevent from landing

Constraints

- $1 \le n \le 10^5$
- $1 \le startHeight[i] \le 10^9$ $(0 \le i < n)$
- $1 \le descentRate[i] \le 10^5$ $(0 \le i < n)$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of planes (the size of *startHeight)*

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *startHeight[i]*

The next line repeats the integer *n,* the size of *descentRate*

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *descentRate[i]*

▼ Sample Case 0

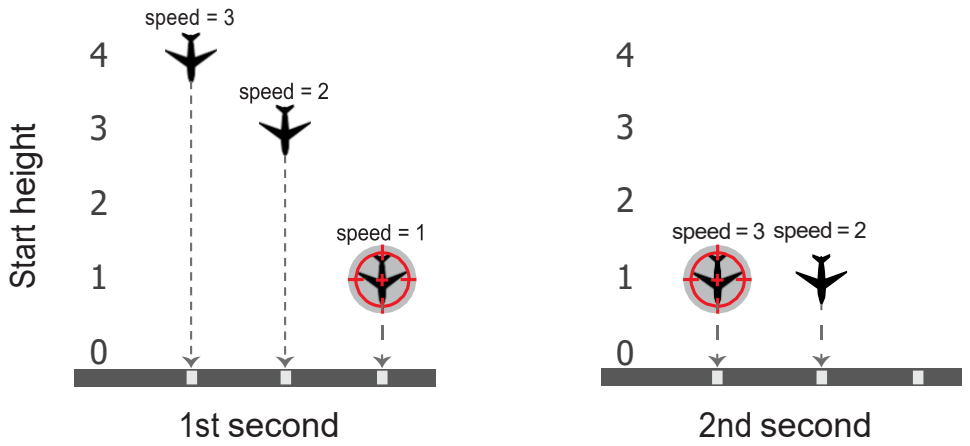
Sample Input

```
STDIN       Function
-------     --------
3       →   startHeight[] size n = 3
4       →   startHeight = [4, 3, 1]
3
1
3       →   descentRate[] size n = 3
3       →   descentRate = [3, 2, 1]
2
1
```

Sample Output

```
2
```

Explanation



There are *3* planes landing.

- Any of the planes can be shot first, but it is best to target the third plane at height 1 descending at a rate of 1. That plane will land before the next second, so this is the only opportunity to fire on it.
- The second shot is fired during second 2. Both of the other planes will land before the third second, so only one can be kept from landing.
- Only 2 planes can be prevented from landing. In the diagram above, the first plane is chosen as a target in the 2nd second.

▼ Sample Case 1

Sample Input

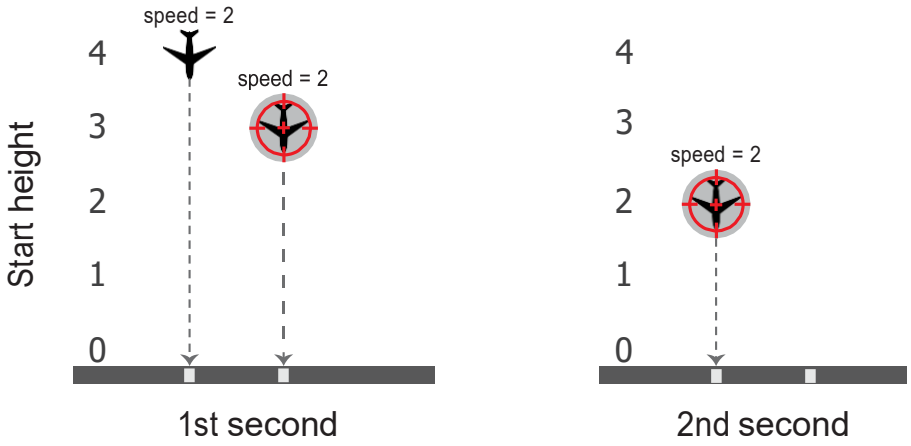
```
STDIN       Function
-------     --------
2       →   startHeight[] size n = 2
```

```
4       →  startHeight = [4, 3]
3
2       →  descentRate[] size n = 2
2       →  descentRate = [2, 2]
2
```

Sample Output

```
2
```

Explanation



The player can shoot both aircraft down, choosing either plane to target first.

Question - 66
Bucket Fill

Digital graphics tools often make available a "bucket fill" tool that will only paint adjacent cells . In one *fill,* a modified bucket tool recolors adjacent cells (connected horizontally or vertically but not diagonally) that have the same color. Given a picture represented as a 2-dimensional array of letters representing colors, find the minimum number of fills to completely repaint the picture.

Example

picture= ["aabba", "aabba", "aaacb"]

Each string represents a row of the picture and each letter represents a cell's color. The diagram below shows the *5* fills needed to repaint the picture. It takes two fills each for *a* and *b*, and one for *c*. The array *picture* is shown below.

Initial Canvas:

a	a	b	b	a
a	a	b	b	a
a	a	a	c	b

Output (No. of Strokes): 5

a	a	b	b	a
a	a	b	b	a
a	a	a	c	b

Stroke 1
Stroke 2
Stroke 3
Stroke 4
Stroke 5

Function Description

Complete the function *strokesRequired* in the editor below.

strokesRequired has the following parameter(s):

string picture[h]: an array of strings where each string represents one row of the picture to be painted

Output:

int: the minimum number of fills required to repaint the picture

Constraints

- *h* and *w* refer to height and width of the graph.
- $1 \le h \le 10^5$
- $1 \le w \le 10^5$
- $1 \le h*w \le 10^5$
- *length(picture[i]) = w (where 0 ≤ i < h)*
- *picture[i][j]* is in the set *{'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

```
STDIN       Function
-------     -----------
3       →   picture[] size h = 3
aaaba   →   picture = [ "aaaba" , "ababa" , "aaaca" ]
ababa
aaaca
```

Sample Output

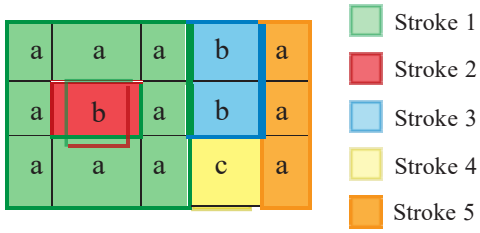
```
5
```

Explanation

Initial Canvas:

a	a	a	b	a
a	b	a	b	a
a	a	a	c	a

Output (No. of Strokes): 5



Letter *a* takes *2* fills, *b* takes *2* fills and *c* takes *1* fill for a total of *5*.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
4       →   picture[] size h = 4
bbba    →   picture = [ "bbba", "abba", "acaa" , "aaac" ]
abba
acaa
aaac
```

Sample Output

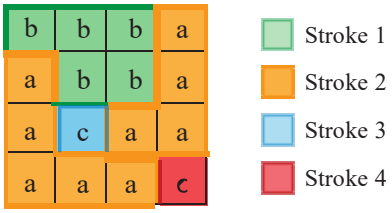
```
4
```

Explanation

Initial Canvas:

b	b	b	a
a	b	b	a
a	c	a	a
a	a	a	c

Output (No. of Strokes): 4



Letters *a* and *b* each take *1* fill and letter *c* takes *2* fills.

Question - 67
Sort an Array

Given an array of integers, any array element can be moved to the end in one move. Determine the minimum number of moves required to sort the array in ascending order.

Example:

arr = [5, 1, 3, 2]

- Move the value *arr[2] = 3* to the end to get *arr = [5, 1, 2, 3]*.
- Move *arr[0] = 5* to the end to achieve the sorted array, *arr = [1, 2, 3, 5]*.
- The minimum number of moves required to sort the array is 2.

Function Description

Complete the function *getMinimumMoves* in the editor with the following parameter:

int arr[n]: an array to sort

Returns

int: the minimum number of moves needed to sort the array in ascending order

Constraints

- $1 \le n \le 10^5$
- $0 \le arr[i] \le 10^6$
- array elements are distinct

▼ Input Format For Custom Testing

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

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

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   arr[] size n = 3
1       →   arr = [1, 2, 3]
2
3
```

Sample Output

```
0
```

Explanation

The array is already sorted, so no moves are necessary.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   arr[] size n = 3
1       →   arr = [1, 3, 2]
3
2
```

Sample Output

```
1
```

Explanation

- Move the value *arr[1] = 3* to the end to get *arr = [1, 2, 3].*
- The minimum number of moves required to sort the array is 1.

Question - 68
Diverse Groups

A professional society is organizing a conference and needs to form a diverse group of 3 members.

The group must meet the following criteria:

- Diversity Rule: The group must include at least one man and one woman.
- Distinct Combinations: Two groups are considered distinct if they have at least one different member.

You are given:

- *m:* The number of eligible men.
- *w:* The number of eligible women.

Your task is to calculate the total number of distinct ways to select a diverse group of 3 members, following the diversity rule.

Example

m = 1

w = 3

There is *m = 1* man available and there are *w = 3* women. Label them *m1*, *w1*, *w2*, *w3* for demonstration. There are *3* possible ways to form a diverse group: (*m1, w1, w2)*, (*m1, w1, w3*) and (*m1, w2, w3*). The only other possible permutation is (*w1, w2, w3*), which does not include a man, so it is invalid.

Function Description

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

int m: the number of men available

int w: the number of women available

Returns

int: the number of ways to select a diverse group from *m* men and *w* women

Constraints

- *0 ≤ m, w ≤ 1000*

▼ Input Format For Custom Testing

The first line contains an integer *m*, that denotes the number of men.

The second line contains an integer *w*, that denotes the number of women.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
3     →   number of men m = 3
0     →   number of women w = 0
```

Sample Output 0

```
0
```

Explanation 0

The number of women is *0* so there is no way to select a diverse group.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-------   -----------
2     →   number of men m = 2
2     →   number of women w = 2
```

Sample Output 1

```
4
```

Explanation 1

In this case, *m = 2* and *w = 2*. This yields *4* ways to select a diverse group: (*m1, w1, w2*), (*m1, m2, w2*), (*m2, w1, w2*), (*m1, m2, w1*).

Question - 69

Frequency of Maximum Value

For a given array of integers, find the maximum value in the segment from each index to the last element of the array. Determine how many times this maximum value appears in each segment. You will be given multiple queries, each representing an index in the array. Create a result array where each element corresponds to the number of times the maximum value occurs in the segment starting at the index specified by each query.

Example

numbers = [5, 4, 5, 3, 2]
q = [1, 2, 3, 4, 5]

Note: The *numbers* array indexes are from *1* to *n* where *n* is the length of the array.

For the first query, the index is 1. The segment starting at index 1 is *[5, 4, 5, 3, 2].* The highest value is *5,* and it occurs *2* times. *result = [2]*
For the second query, the index is 2. The segment starting at index 2 is *[4, 5, 3, 2].* The highest value is *5,* and it occurs *1* time. *result = [2, 1]*
In each of the remaining segments queried, *[5, 3, 2], [3, 2],* and *[2],* there is only one occurrence of a highest value so a 1 is appended to *result* for each query.

The final array returned is *[2, 1, 1, 1, 1].*

Function Description

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

int[n] numbers: the array to analyze
int[m] q: an integer array that contains the index values for each query

Returns

int[m]: an integer array with the answers to each query, aligned by index

Constraints

- $1 \le n, m \le 10^5$
- $1 \le numbers[i] \le 10^6$
- $1 \le q[j] \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in the *numbers* array.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer describing *numbers[i]*.
The next line contains an integer, *m*, denoting the number of elements in the query array, *q*.
Each line *j* of the *m* subsequent lines (where $0 \le j < m$) contains an integer describing *q[j]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   numbers[] size n = 3
2       →   numbers = [2, 1, 2]
1
2
3       →   q[] size m = 3
1       →   q = [1, 2, 3]
2
3
```

Sample Output

```
2
1
1
```

Explanation

There are 3 queries.

- For the first query, find the frequency of the maximum value in the *numbers* array starting from index *1* up to index *3*. The maximum value from index *1* to index *3* is *2* and its frequency is *2*. So, the answer is *2*.
- The second query denotes that you have to find the frequency of the maximum value in the *numbers* array starting from index *2* up to index *3*. The maximum value from index *2* to index *3* is 2 and its frequency is *1*. So, the answer is *1*.
- The third query denotes that you have to find the frequency of the maximum value in the *numbers* array starting from index *3* up to index *3*. The maximum value from index *3* to index *3* is 2 and its frequency is *1*. So, the answer is *1*.
- Return the integer array *result = [2, 1, 1]*

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   numbers[] size n = 3
2       →   numbers = [2, 2, 2]
2
2
3       →   q[] size m = 3
1       →   q = [1, 2, 3]
2
3
```

Sample Output

```
3
2
1
```

Explanation

There are 3 items in the *numbers* array with values *2*, *2* and *2*. There are 3 queries.

- The first query denotes that you have to find the frequency of the maximum value in the *numbers* array starting from index *1* up to index *3*. The maximum value from index *1* to index *3* is 2 and its frequency is *3*. So, the answer is *3*.
- The second query denotes that you have to find the frequency of the maximum value in the *numbers* array starting from index *2* up to index *3*. The maximum value from index *2* to index *3* is 2 and its frequency is *2*. So, the answer is *2*.
- The third query denotes that you have to find the frequency of the maximum value in the *numbers* array starting from index *3* up to index *3*. The maximum value from index *3* to index *3* is 2 and its frequency is *1*. So, the answer is *1*.
- Return the integer array *result = [3, 2, 1]*

Question - 70

Game Segments

A video game developer is creating a game where the character navigates through several segments of a level. In each segment, if the character collects a coin, the player earns a point. If there is no coin in a segment, the player loses a point.

Player 1 always starts the level, and at a certain point, the gameplay is handed over to Player 2 to finish the level. Player 1 aims to have a higher score than Player 2 by the end of the level.

Given the status of each segment (whether it contains a coin or not), determine the minimum number of segments Player 1 should play to ensure their score is higher than Player 2's score at the end.

Example

segments = [1, 1, 0, 1]

Player 1 has the following options:

- Play *0* segments. This gives them a score of *0*. Player *2* gets a score of *3 - 1 = 2* (because they gain a point for each of the *3* segments with a coin, and lose *1* point for the segment without a coin).
- Play *1* segment. This gives them a score of *1*. Player *2* gets a score of *2 - 1 = 1*.
- Play *2* segments. This gives them a score of *2*. Player *2* gets a score of *1 - 1 = 0*.

Only in the last case, by playing *2* segments, does Player 1's score beat Player 2's. Therefore, return the answer *2*.

Function Description

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

int coins[n]: denotes whether a video game segment contains a coin (1) or not (0)

Returns:

int: the minimum number of segments Player 1 must play so that their score is greater than Player 2's score

Constraints

- $1 \le n \le 10^5$
- *coins[i]* is either 1 or 0

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *coins[]*.

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

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     --------
```

```
5       →  coins[] size n = 5
1       →  coins = [1, 0, 0, 1, 0]
0
0
1
0
```

Sample Output

```
0
```

Explanation

If Player 1 plays 0 segments, their score is 0, and Player 2's score is 2 - 3 = -1.

▼ Sample Case 1

Sample Input

```
STDIN       Function
--------    --------
5       →  coins[] size n = 5
1       →  coins = [1, 1, 1, 0, 1]
1
1
0
1
```

Sample Output

```
2
```

Explanation

- If Player 1 plays 0 segments, their score is 0, and Player 2's is 4 - 1 = 3.
- If Player 1 plays 1 segment, their score is 1, and Player 2's is 3 - 1 = 2.
- If Player 1 plays 2 segments, their score is 2, and Player 2's is 2 - 1 = 1.

Question - 71
Office Design

A company is planning to repaint its office and wants to select colors that complement each other. They have various color options available at different prices, and these colors are listed in an order where adjacent colors complement each other. Given the company's budget, determine the maximum number of consecutive colors they can purchase within their budget.

Example

prices = [2, 3, 5, 1, 1, 2, 1]
money = 7

All subarrays that sum to less than or equal to 7:
Length 1 subarrays are [2], [3], [5], [1], [1], [2], [1]
Length 2 - [2, 3], [5, 1], [1, 1], [1, 2], [2, 1]
Length 3 - [5, 1, 1], [1, 1, 2], [1, 2, 1]
Length 4 - [1, 1, 2, 1]

The longest of these, or the maximum number of colors that can be purchased, is 4.

Function Description

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

int prices[n]: the prices of the various paint colors

int money: the amount of money the company can spend on paints

Returns

int: the maximum number of colors the company can purchase

Constraints

- $1 \le n \le 10^5$
- $1 \le prices[i] \le 100$
- $1 \le money \le 10^6$

▼ Input Format For Custom Testing

The first line contains an integer, *n*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer describing *prices[i]*.
The next line contains an integer, *money*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   prices[] size n = 3
10      →   prices = [ 10, 10, 10 ]
10
10
5       →   money = 5
```

Sample Output

```
0
```

Explanation

There are 3 colors to choose from, each costing 10. With the company budget of 5, no colors can be purchased.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   prices[] size n = 3
5       →   prices = [ 5, 10, 10 ]
10
10
5       →   money = 5
```

Sample Output

```
1
```

Explanation

With the company budget of 5, only the first color can be purchased.

Question - 72

Maximum Occurring Character

Create a function that:

1. Accepts a string as input

2. Identifies the character that appears most frequently in the string
3. Returns this character

Requirements:

- The string will contain only ASCII characters from the ranges ('a'-'z', 'A'-'Z', '0'-'9').
- The function must be case-sensitive ('A' and 'a' are different characters).
- If there is a tie for the most frequent character, return the one that appears first in the string.

Example

findMostFrequentChar("abcABCabc") should return 'a'
findMostFrequentChar("1223334444") should return '4'

In the first example, characters 'a', 'b', and 'c' occur twice, and 'a' appears first.

Function Description

Complete the function *maximumOccurringCharacter* in the editor with the following parameter:

string text: the string to analyze

Returns

char: the most common character that appears first in the string

Constraints

- $10 \leq$ length of text $\leq 10^4$

▼ 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 *3* times.

▼ 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, it is the answer.

Question - 73
Slanted Cipher

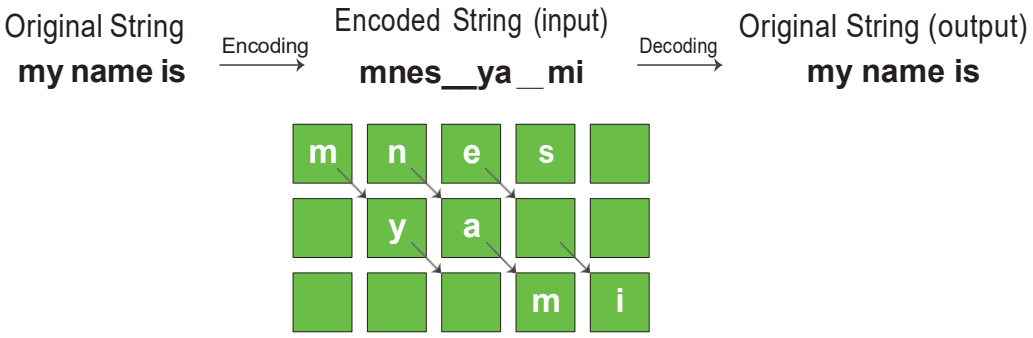
The slanted transposition cipher encodes text by arranging characters in a grid-like structure and filling them diagonally, row by row.

Example

string = 'my name is'

number of rows = 3

The string 'my name is' is encoded as 'mnes__ya ____ mi'.



After filling the grid, the encoded string is formed by concatenating the characters row by row, with any remaining spaces filled with underscores ('_').

Encoding Process

1. Initialize an empty grid with *numberOfRows* rows.
2. Starting from the top left corner, write characters diagonally, moving downward and right. When you reach the last row, wrap back to the top.
3. If the input string is shorter than the grid's capacity, fill the remaining spaces with underscores (`'_'`).
4. Read each row sequentially and concatenate characters to form the encoded string.

The original string can be reconstructed by reversing this process. Given the number of rows used and the encoded string, find and return the decoded string.

Note: The number of rows will always be less than the length of the original encoded string. Any trailing spaces can be ignored. The final output should be read row-wise.

Function Description

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

int numberOfRows: the number of rows used in the encoding process

string encodedString: the encoded string

Returns

string: the decoded string

Constraints

- $1 \le numberOfRows \le 2 \times 10^3$
- $1 \le$ size of *encodedString* $\le 2 \times 10^6$
- *'a'* $\le$ *encodedString[i]* $\le$ *'z' or encodedString[i]* = *'_'*, where $0 \le i <$ size of *encodedString*
- The length of the encoded string is always a multiple of the number of rows.

▼ Input Format For Custom Testing

The first line contains an integer, *numberOfRows*.

The next line contains a string, *encodedString*.

▼ Sample Case 0

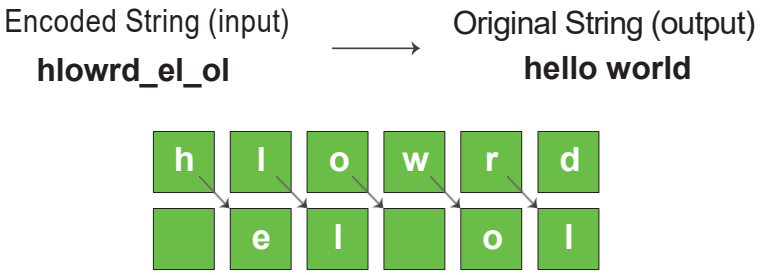
Sample Input

```
STDIN           Function
-------         --------
2            → numberOfRows = 2
hlowrd_el_ol → encodedString = 'hlowrd_el_ol'
```

Sample Output

```
hello world
```

Explanation



After decoding the given *encodedString*, return the decoded string, 'hello world'.

▼ Sample Case 1

Sample Input

```
STDIN               Function
-----               --------
3                → numberOfRows = 3
mnes__ya_____mi → encodedString = 'mnes__ya_____mi'
```

Sample Output

```
my name is
```

Explanation

Refer to the image given in the problem statement. After decoding the given *encodedString,* return the decoded string 'my name is'.

Question - 74

Video Buffering

A video streaming service sends *n* numbered packets that arrive at a rate of *arrivalRate* packets per second. The packets do not arrive all at once. The video player processes these packets as follows:

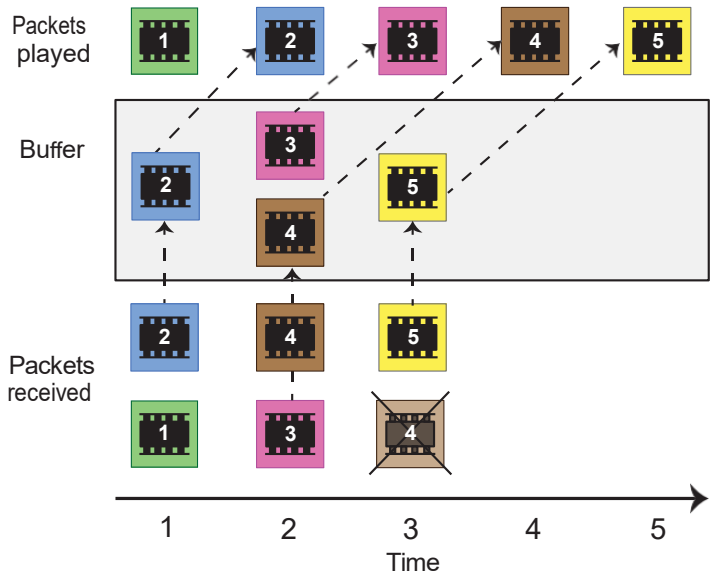
1. The video player plays one packet per second in numerical order, starting from packet 1 at time *t* = 1.
2. Packets that may be used in the future are stored in a buffer, while packets that have already been played are discarded. Duplicate packets can be ignored.
3. The video player starts "re-buffering" whenever the correct packet is not immediately available, either from the buffer or as the first element received from the data stream at that time.
4. If multiple packets arrive at a particular second *t*, only the first packet is played (either from the buffer or from the arrival), and the remaining packets will be buffered or ignored.

Your task is to find the first time at which the video begins re-buffering. If re-buffering is never necessary, return -1. The video ends just after the end of the array is reached.

Example

arrivalRate = 2

packets = [1, 2, 3, 4, 4, 5]



At time $t = 1$:

- Packets 1 and 2 are received
- Packet 1 is played
- Packet 2 is buffered

At time $t = 2$:

- Packet 2 is played from the buffer
- Packets 3 and 4 are received and buffered

At time $t = 3$:

- Packet 3 is played from the buffer
- A duplicate packet 4 is received and ignored
- Packet 5 is received and buffered

The stream can continue to be played from the buffer, and there is no time at which a packet is not available whose number matches the time. Therefore, the return value is -1.

Function Description

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

int arrivalRate: the rate of arrival of packets

int packets[n]: each *packets[i]* denotes the packet number of the i^{th} packet that arrives

Returns

int: the first time that re-buffering occurs or -1

Constraints

- $1 \le arrivalRate \le 50$
- $1 \le n \le 10^6$
- $1 \le packets[i] \le 10^6$

▼ Input Format For Custom Testing

The first line contains an integer, *arrivalRate*, denoting the rate of arrival of packets.

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

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

▼ Sample Case 0

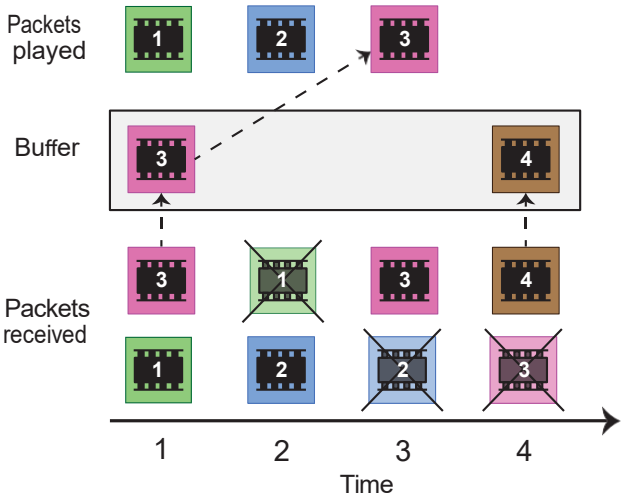
Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
2     →     arrivalRate = 2
8     →     packets[] size n = 8
1     →     packets = [1, 3, 2, 1, 2, 3, 3, 4]
3
2
1
2
3
3
4
```

Sample Output

```
4
```

Explanation



As shown in the image:

- Packets 1 and 3 arrive beginning at $t = 1$: Packet 1 is played in the first second, and packet 3 is placed in the buffer.
- Packets 2 and 1 arrive beginning at $t = 2$: Packet 2 is played, and packet 1 is ignored. Packet 3 remains in the buffer.
- Packets 2 and 3 arrive at $t = 3$: Packet 2 is ignored and packet 3 is played from the buffer. The duplicate packet 3 is then ignored.
- Packets 3 and 4 arrive at $t = 4$: Packet 3 is ignored, and packet 4 is not available in the buffer. Packet 4 is required at time $t = 4$ but arrives after 3, so it is not available exactly at time 4. Hence, re-buffering occurs at time $t = 4$.

▼ Sample Case 1

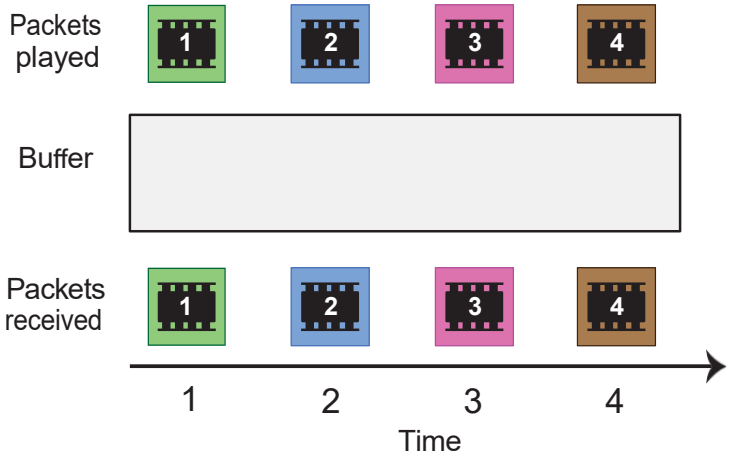
Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
1     →     arrivalRate = 1
4     →     packets[] size n = 2
1     →     packets = [1, 2, 3, 4]
2
3
4
```

Sample Output

```
-1
```

Explanation



No buffering occurs in this case, as each packet is available from the stream when needed.
At *t*=1, packet 1 is required, and it is present as it just arrives at *t*=1. The same reasoning is applied for packets 2, 3, and 4.

Question - 75
Process Tree

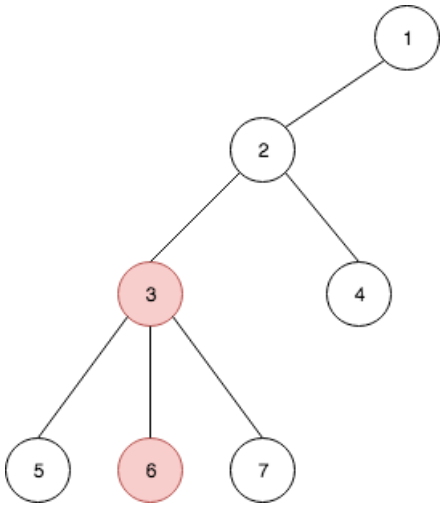
A sequence of processes is represented as a tree structure. Each process is numbered sequentially starting from 1. Process number *n* spawns *n* new child processes. For example:

- Process 1 spawns 1 child process.
- Process 2 spawns 2 child processes.
- Process 3 spawns 3 child processes.

Given a specific process number, determine the process number of its parent.

Example
processNumber = 6

From the diagram, the parent of *6* is *3*.



Function Description
Complete the *findParent* function in the editor with the following parameter(s):
int processNumber: the process number to query

Returns

int: the process number of the parent

Constraints

- $2 \leq processNumber \leq 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *processNumber*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     -----------
6       →   processNumber = 6
```

Sample Output

```
3
```

Explanation

Refer to the graph in the problem statement. The child of *1* is *2*. Children of *2* are *3* and *4*. The children of *3* are *5, 6,* and *7*. Therefore, the process number of the parent process of the given process is *3*.

Question - 76

Hashed Ports

Packets are sent to ports based on a hash value calculated as *hash = packet_id % numberOfPorts* where % represents the modulo division operator. Ports are numbered from 0 to (*numberOfPorts* - 1). Each port requires *transmissionTime* seconds to process a packet. If a port is busy processing a packet when a new one arrives, the new packet is rerouted to the next port number in a circular manner.

Given a list of packet IDs that arrive one per second, determine the final port to which each packet is sent.

Example

numberOfPorts = 3
transmissionTime = 2
packetIds = [4, 7, 10, 6]

Return: [1, 2, 1, 0]

Explanation:

- Packet 4 arrives at *t* = 1, *hash*= 4 % 3 = 1, sent to port 1
- Packet 7 arrives at *t* = 2, *hash*= 7 % 3 = 1, but port 1 is busy until t=3, so sent to port 2
- Packet 10 arrives at *t* = 3, *hash*= 1 0 % 3 = 1, port 1 is now free, so sent to port 1
- Packet 6 arrives at *t* = 4, *hash*= 6 % 3 = 0, sent to port 0

Function Description

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

int numberOfPorts: the number of ports in the system
int transmissionTime: the time for a port to send a packet
int packetIds[n]: the IDs of the packets in the order in which they arrive

Returns:

int[n]: the ports to which the packets are sent

Constraints

- $1 \le numberOfPorts \le 2000$
- $1 \le transmissionTime \le 100$
- $1 \le n \le 2000$
- $1 \le packetIds[i] \le 10^5$

▼ Input Format For Custom Testing

The first line contains an integer, *numberOfPorts*, the number of ports in the system.
The next line contains an integer, *transmissionTime*, the sending time for each port.
The next line contains an integer, *n,* the number of packets and size of *packetIds[]*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *packetIds[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
4       →   numberOfPorts = 4
2       →   transmissionTime = 2
3       →   packetIds[] size n = 3
0       →   packetIds = [0, 2, 6]
2
6
```

Sample Output

```
0
2
3
```

Explanation

According to the hashes, packets should be sent on ports 0, 2, and 2 respectively. but packet 6 arrives at time 3, at which packet 2 is in the process of being sent. Therefore, it is sent to the next port. Return the array [0, 2, 3].

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
5       →   numberOfPorts = 5
3       →   transmissionTime = 3
4       →   packetIds[] size n = 4
1       →   packetIds = [1, 6, 11, 16]
6
11
16
```

Sample Output

```
1
2
3
1
```

Explanation

- Packet 1 is sent to port 1 at time $t = 1$, so it will be processed until time $t = 3$.
- Packet 6 is sent to port 1 at time $t = 2$, but port 1 is processing packet 1, so it is sent to port 2.
- Packet 11 is sent to port 1 at time $t = 3$, but port 1 is processing packet 1 and port 2 is processing packet 6, so it is sent to port 3.
- Packet 16 is sent to port 1 at time $t = 4$, so port 1 is free to process packet 16.

Therefore, the packets are sent to ports 1, 2, 3, and 1, respectively.

Question - 77
Authentication Tokens

When users are authenticated, they receive an authentication token. This token expires after a system-wide *expiryLimit* unless it is reset. If a reset request is made on or before the expiry time, the expiry is extended by *expiryLimit* minutes from the reset time.

1. An unexpired *token_id* can be reset any number of times.
2. A reset issued to a non-existent or an expired *token_id* is ignored.
3. Once a *token_id* expires it cannot be reused.

Command syntax: `[type, token_id, T]`

- *Create command*: Type 0 generates a token with id *token_id* at time *T*. Its expiry is set to *T + expiryLimit*.
- *Reset command:* Type 1 resets the expiry to *T + expiryLimit*.

Start with an empty list of tokens. Perform a sequence of requests sorted ascending by their *T* parameter. Find the number of tokens that are active after all commands have been executed, at the maximum *T* of all requests.

Example
expiryLimit = 4
commands = [[0,1,1], [0,2,2], [1,1,5],[1,2,7]]

The maximum time *T = 7*, so the analysis will end at *T = 7*. Each time a token is created or reset, its new expiration time will be at time *T + 4*.
Working through the commands:

0. *[0,1,1]:* Create *token_id = 1* at time *T = 1* and set its expiry to *T + expiryLimit = 5*.
1. *[0,2,2]:* Create *token_id = 2* with an expiry at *T = 6*
2. *[1,1,5]:* Reset *token_id = 1* at *T = 5*. The time is less than or equal to the expiry limit so a new limit is set: *5 + 4 = 9*.
3. *[1,2,7]:* Reset *token_id = 2* at *T = 7*. The id expires at time *T = 6*, so when the *Reset token_id = 2* command comes in at *T = 7*, it is ignored.

Only *token_id = 1* is active at time *T = 7*. Return 1.

Function Description

Complete the *numberOfTokens* function in the editor with the following parameter(s):
int expiryLimit: the expiry limit of each token
int commands[n][3]: each *commands[i],* has *3* integers: *[command, token_id, T]*.

Returns
int: the number of tokens that exist at the end of the command stream

Constraints
- The *commands* array is given sorted ascending by *T (commands[i][2])*.
- $1 \le expiryLimit < 10^8$
- $1 \le n < 10^5$
- $1 \le token\_id < 10^8$
- $1 \le T < 10^8$

▼ Input Format For Custom Testing

The first line contains the integer, *expiryLimit*.
The second line contains the integer *n*, that denotes the number of elements in *commands*.

The third line contains the integer *3*, that denotes the number of details required to describe each command.
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n)* contains three space-separated integers that represent *commands[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
3       →   expiryLimit = 3
2       →   commands[] size n = 2
3       →   commands[i][] size = 3 (always)
0 1 1   →   commands = [[0, 1, 1], [1, 1, 5]]
1 1 5
```

Sample Output

```
0
```

Explanation
The token with *id 1* is created at time *1*. It expires at time *4*. The reset command at time *5* is ignored since the token has expired. There are no tokens left.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
3
3
0 1 1
1 1 4
1 2 5
```

Sample Output

```
1
```

Explanation
One token *1* is created at time *1*. It is supposed to expire at time *T = 4*, but it is reset at *T = 4*. Its next expiry is at *T = 7*. Since token *2* does not exist, the reset command *1 2 5* does not have any effect. There is one token at the end when *T = 5*.

Question - 78
Document Chunking

A financial services company uploads documents to a compliance system using a chunking mechanism:

- Each document is divided into equal-sized packets numbered from 1 to *totalPackets*.
- Documents are divided and uploaded in "chunks" of packets, where a chunk is a contiguous collection of 2^n packets (*n* is any integer ≥ 0).
- Randomly selected chunks are uploaded until the entire document is completely uploaded.

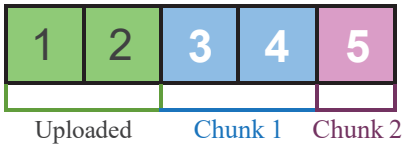
Given a partially uploaded document described in *uploadedChunks*, determine the minimum number of chunks yet to be uploaded.

Example
totalPackets = 5
n = 2 (size of *uploadedChunks*, i.e., the number of chunks already uploaded)
uploadedChunks = [[1, 2]]

The document has 5 packets, and 1 chunk of $2^1 = 2$ packets [1, 2] is already uploaded.
The remaining 3 packets must be uploaded in 2 chunks:

- Chunk 1: $2^1 = 2$ packets [3, 4]
- Chunk 2: $2^0 = 1$ packet [5]

So the minimum number of chunks yet to be uploaded is 2.



Function Description

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

long int totalPackets: the number of packets in the document.

long int uploadedChunks[n][2]: each *uploadedChunks[i]* describes the start and end packet numbers of the uploaded chunks.

Returns

int: the minimum number of chunks that need to be uploaded

Constraints

- $1 \le totalPackets < 10^{18}$
- $0 \le$ length of $uploadedChunks < 10^5$
- The uploaded chunks do not overlap.
- $1 \le uploadedChunks[i][0] \le uploadedChunks[i][1] \le totalPackets$

▼ Input Format For Custom Testing

The first line contains a positive integer, *totalPackets*.

The next line contains an integer, *n*, the number of chunks already uploaded and the size of the major array, *uploadedChunks*.

The next line contains an integer, *m = 2,* denoting the number of columns in each *uploadedChunks[i]*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains two space-separated integers representing the starting and ending packet numbers of each of the already uploaded chunks.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          Function
-----          --------
10         →   totalPackets = 10
2          →   uploadedChunks[][] size n = 2
2          →   uploadedChunks[][] size m = 2 (always)
1 2        →   uploadedChunks = [[ 1, 2 ] , [ 9 , 10 ]]
9 10
```

Sample Output

```
2
```

Explanation

- The document has 10 packets and 2 chunks of $2^1 = 2$ packets are already uploaded [1, 2] and [9, 10].
- The remaining 10-4 = 6 packets are uploaded in chunks. The length of chunk1 is $2^2 = 4$, packets [3,4,5,6], and the length of chunk2 is $2^1 = 2$, packets [7, 8].

▼ Sample Case 1

Sample Input For Custom Testing

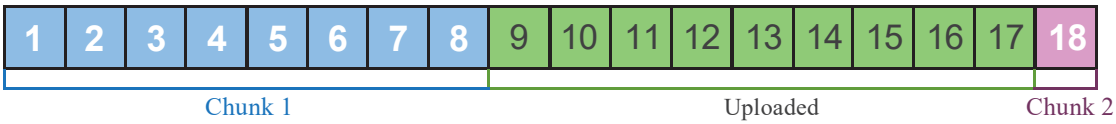
```
STDIN          Function
-----          --------
18         →   totalPackets = 18
1          →   uploadedChunks[][] size n = 1
```

```
2          →   uploadedChunks[][] size m = 2
9 17       →   uploadedChunks = {{}}
```

Sample Output

```
2
```

Explanation



- The document has 18 packets and 2 chunks of packets are already uploaded: [9, 10, 11, 12, 13, 14, 15, 16] and [17] .
- The remaining 18-2 = 16 packets are uploaded in chunks. The length of chunk1 is $2^3 = 8$ packets: [1, 2, 3, 4, 5, 6, 7, 8] and the length of chunk2 is $2^0 = 1$ packets: [18].

Question - 79
Who Is the closest?

Given a string *s* and a list of queries, for each *query[i]* (where $0 \le query[i] <$ length of *s*), find the index of the closest matching character (same character as the one at position *query[i]*). If there are multiple matching characters at equal distances, print the lower index. If no matching character exists, print -1.

Example

s = "babab"

queries = [2]

The letter at *s[2]* is 'b'. Two other 'b' characters are at positions 0 and 4, both at distance 2. Since we choose the lower index when there is a tie, the answer is 0.

Function Description

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

string s: the original string

int queries[n]: each *queries[i]* is an index of a character

Prints

Print the answers to the queries, each on a new line. There is no return value.

Constraints

- length of *s*, *queries[i]* $\le 10^5$
- $1 \le n \le 10^5$
- *s* will contain only lowercase English letters, ascii[a-z]

▼ Input Format For Custom Testing

The first line contains a string, *s*.

The second line contains an integer, *n*, the size of *queries*.

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

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
hackerrank  s = "hackerrank"
4           queries[] size n = 4
4           queries = [4, 1, 6, 8]
```

```
1
6
8
```

Sample Output 0

```
-1
7
5
-1
```

Explanation 0

Query #0: Character at index 4 is 'e'. There is no other 'e' present in s, so we print -1.
Query #1: Character at index-1 is 'a'. There is only one closest index (index-7) that contains 'a'.
Query #2: Character at index-6 is 'r'. There is only one closest index (index-5) that contains 'r'.
Query #3: Character at index-8 is 'n'. There is no other 'n' present in *s*, so we print -1.

▼ Sample Case 1

Sample Input 1

```
aaaa
4
0
1
2
3
```

Sample Output 1

```
1
0
1
2
```

Explanation 1

Query #0: Character at index-0 is 'a'. There is only one closest index (index-1) that contains 'a'.
Query #1: Character at index-1 is 'a'. There are two closest indexes (index-0 and index-2) that also contain 'a'. Since 0 is smaller than 2 we print 0.
Query #2: Character at index-2 is 'a'. There are two closest indexes (index-1 and index-3) that also contain 'a'. Since 1 is smaller than 3 we print 1.
Query #3: Character at index-3 is 'a'. There is only one closest index (index-2) that contains 'a'.

▼ Sample Case 2

Sample Input 2

```
sam
1
1
```

Sample Output 2

```
-1
```

Explanation 2

Query #0: Character at index-1 is 'a'. There is no other 'a' present in *s* so we print -1.

Question - 80
Largest XOR

Given a binary string *x*, generate another binary string *y* of the same length that maximizes the result when XORed with *x*. The constraint is that the number of bits set to 1 in *y* cannot exceed *maxSet*. Return the binary string *y*.

Note: The binary strings will always have *bits* digits (an integer), and leading zeros are fine.
Example

bits = 3
maxSet = 1
x = 101

First, determine all possible *bits = 3* digit binary strings with only *maxBits = 1* or fewer bits set: *000, 001, 010, 100.* These are the potential *y* values.

1. Now, XOR each of the *y* values with *x = 101*
    1. 000 XOR 101 = 101
    2. 001 XOR 101 = 100
    3. **010 XOR 101 = 111**
    4. 100 XOR 101 = 001

The third value produces the maximal result, where *y* = 010. Return the string '010'.

Function Description

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

int bits: the length of the binary strings *x* and *y*
int maxSet: the number of bits that may be set in *y*
string x: a binary string

Returns

string: the best *y* value as a binary string

Constraints

- $1 \leq maxSet \leq bits \leq 10^5$
- The string *x* contains only *0*s and *1*s.

▼ Input Format For Custom Testing

The first line contains an integer, *bits*.
The second line contains an integer, *maxSet*.
The third line contains a string, *x*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4      →    bits = 4
1      →    maxSet = 4
1011   →    x = "1011"
```

Sample Output

```
0100
```

Explanation

1. All of the values of *y* with *bits = 4* digits and *maxSet = 1* or fewer bits set are *0000, 1000, 0100, 0010, 0001*,
2. XOR each of those values with *x = 1011*
    1. 0000 XOR 1011 = 1011
    2. 1000 XOR 1011 = 0011
    3. **0100 XOR 1011 = 1111**
    4. 0010 XOR 1011 = 1001
    5. 0001 XOR 1011 = 1010

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
----------  -----------
3        → bits = 3
2        → maxSet = 2
001      → x = "001"
```

Sample Output

```
110
```

Explanation

1. All of the values of *y* with *bits = 3* digits and *2* bits set are *110, 101, 011.*
2. All of the values of *y* with *bits = 3* digits and *1* bit set are *100, 010, 001.*
3. Also analyze the identity value *000*, as always.
4. XOR each of those values with *x = 001*
    1. **110 XOR 001 = 111**
    2. 101 XOR 001 = 100
    3. 011 XOR 001 = 010
    4. 100 XOR 001 = 101
    5. 010 XOR 001 = 011
    6. 001 XOR 001 = 000
    7. 000 XOR 001 = 001

Question - 81
Vowels

Given an array of strings containing *n* elements, each consisting of lowercase English letters, and *q* queries in the format *l-r*, for each query, determine how many strings from index *l* to *r* have both the first and last character as vowels.

Vowels are defined as {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 with the following parameters:
str strArr[n]: an array of strings
str query[q]: an array of strings, each of which describes an interval *l-r* using integers delimited by a dash

Returns
int[q]: the results of each query

Constraints
- $1 \le n, q \le 10^5$
- $1 \le l \le r \le n$
- *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 $1 \leq i \leq n$) 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 $0 \leq j < q$) 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 q = 2
2-3      →     query = [ "2-3", "4-5" ]
4-5
```

Sample Output

```
1
1
```

Explanation

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 q = 2
1-2      →     query = [ "1-2", "2-3" ]
2-3
```

Sample Output

```
1
2
```

Explanation

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.

Question - 82
Break a Palindrome

A palindrome reads the same forwards and backwards, like "mom". Modify a palindrome by changing exactly one character to another character within the ASCII range [a-z]. The goal is to ensure the new string fulfills the following criteria:

1. It is not a palindrome.
2. It is alphabetically lower than the original palindrome.

3. It is the smallest possible string alphabetically that can be obtained by changing just one character.

Return the new string, or "IMPOSSIBLE" if it is not feasible to create such a string.

Example

palindromeStr = 'aaabbaaa'

- Possible strings lower alphabetically than *'aaabbaaa'* after one change are *['aaaabaaa', 'aaabaaaa']*.
- *'aaaabaaa'* is not a palindrome and is the lowest string that can be created from *palindromeStr*.

Function Description

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

string palindromeStr: the original string

Returns:

string: the resulting string, or *IMPOSSIBLE* if one cannot be formed

Constraints

- 1 ≤ length of *palindromeStr* ≤ 1000
- *palindromeStr* is a palindrome
- *palindromeStr* contains only lowercase English letters

▼ Input Format For Custom Testing

Locked stub code in the editor reads a single string, *palindromeStr,* from stdin and passes it to the function.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
bab     →   palindromeStr  = 'bab'
```

Sample Output

```
aab
```

Explanation

- Possible strings lower alphabetically than *'bab'* after one change are *['aab', 'baa']*.
- *'aab'* is not a palindrome and is the lowest string that can be created from *palindromeStr*.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
aaa     →   palindromeStr  = 'aaa'
```

Sample Output

```
IMPOSSIBLE
```

Explanation

- There are no strings lower alphabetically than *'aaa'* after one change.
- So, it is IMPOSSIBLE to create a string lower than *'aaa'*.

▼ Sample Case 2

Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
acca    →   palindromeStr  = 'acca'
```

Sample Output

```
aaca
```

Explanation

- Possible strings lower alphabetically than *'acca'* after one change are *['abca', 'aaca', 'acba', 'acaa']*.
- *'aaca'* is not a palindrome and is the lowest string that can be created from *palindromeStr*.

Question - 83
Valid Email Addresses

Valid HackerRank email addresses are of the form "user@hackerrank.com", where the user portion must follow these rules:

- Starts with 1 to 6 lowercase English letters [a-z]
- Followed by an optional underscore character '_' (zero or one occurrence)
- Followed by 0 to 4 optional digits [0-9]

Complete the code by replacing the blank with a regular expression that matches valid email addresses according to these criteria.

Example

Valid email: "abcdef_1234@hackerrank.com"

Invalid email: "a1_1@baddomain.com" (digits cannot precede the underscore, and the domain is not "hackerrank.com")

Constraints

- $1 \leq query \leq 10^3$
- $1 \leq$ string length $\leq 10^3$.

▶ Input Format

▼ Sample Case 0

Sample Input 0

```
STDIN                    Function
-----                    --------
5                      → query = 5
robin@hackerrank.com   → each of these strings is queried via regex
robin_@hackerrank.com
robin_0@hackerrank.com
robin0_@hackerrank.com
robin@gmail.com
```

Sample Output 0

```
True
True
True
False
False
```

Explanation 0

The following *query = 5* validations are performed:

1. "robin@hackerrank.com" starts with betwee*n 1 and 6 lowercase letters and contains zero of the optional characters, so it is valid.*
2. "robin_@hackerrank.com" starts with between 1 and 6 lowercase letters, is followed by a single underscore, and contains none of the optional digits, so it is valid.
3. "robin_0@hackerrank.com" starts with between 1 and 6 lowercase letters, is followed by a single underscore, and is followed by between 0 and 4 digits, so it is valid.
4. "robin0_@hackerrank.com" has valid lowercase letters followed by a valid digit, but the digit must not precede the underscore.
5. "robin@gmail.com" has a valid user name, but its domain and extension do not match "hackerrank.com".

Question - 84
Large Responses

You are given a log file with a list of GET requests delimited with double quotes and spaces.

A sample and the structure of the text file containing the log entries are given below.

Sample log record:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
```

Log file structure:

Hostname	-	-	Timestamp	Request	HTTPResponseCode	Bytes
unicomp6.uni	-	-	[01/Jul/199	"GET/shut	200	3985

comp.net			5:00:00:06-0400]	tle/countdown/HTTP/1.0"		

Given a ***filename*** that denotes a text file in the current working directory. Create an output file with the name "bytes_" prefixed to the filename (bytes_filename) which stores the information about large responses.

Example: *filename = "hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *bytes_hosts_access_log_00.txt.*

Write the following to the output file:

1. The first line must contain the number of requests that have more than *5000* bytes sent in their response.
2. The second line must contain the total sum of bytes sent by all responses sending more than *5000* bytes.

Note:
The output file has to be written to the current directory.

Constraints

- It is guaranteed that the total number of bytes sent by all large responses does not exceed 10^{12}.
- The log file contains no more than 2×10^5 records.

▼ Input Format for Custom Testing

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

The only line contains a string *filename*, the name of the log file.

▼ Sample Case 0

Sample Input 0

```
STDIN                          Function
-----                          --------
hosts_access_log_00.txt     →  filename = "hosts_access_log_00.txt"
```

Sample Output 0
Given *filename = "hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *bytes_hosts_access_log_00.txt* that contains the following rows:

```
2
```

```
80620
```

Explanation 0

The log file *hosts_access_log_00.txt* contains the following log records:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

The data contains the following large responses:

1. The sixth log record sent a large response:

Hostname	-	-	Timestamp	Request	HTTPResponseCode	Bytes
unicomp6.unicomp.net	-	-	[01/Jul/1995:00:00:14 -0400]	"GET /shuttle/countdown/count.	200	40310

				gifHTTP/1.0"		

Observe that *40310 > 5000*.

2. The ninth log record sent a large response:

Hostname	-	-	**Timestamp**	**Request**	**HTTPResponseCode**	**Bytes**
d104.aa.net	-	-	[01/Jul/1995:00:00:15-0400]	"GET/shuttle/countdown/count.g	200	40310

Observe that *40310 > 5000*.

Because there are a total of two records that sent a total of *40310 + 40310 = 80620* bytes in their collective responses, the first line of our output file is *2* and the second line of our output file is *80620*.

Question - 85

Find the Winner!

Your task is to determine the winner of a card game played between Andrea and Maria.

Game rules:

1. Each player has a deck of numbered cards in a face-down pile.
2. At the beginning, someone calls "Even" or "Odd".
3. If "Even" is called, both players flip their top cards.
4. If "Odd" is called, both players discard their top cards, then flip the next cards.
5. After flipping, Andrea subtracts Maria's card value from her own and adds the result to her score.
6. Maria subtracts Andrea's card value from her own and adds the result to her score.
7. For subsequent rounds, players alternately discard then flip cards.
8. Once all cards are played, the player with the most points wins.

Example

Maria's cards (face down): *andrea* = [3, 5, 6]
Andrea's cards (face down): *maria* = [4, 5, 7]
Starting call: "Even"

```
Maria's Andrea's        Maria's         Andrea's
Card    Card            Score           Score
3       4               3 - 4 = -1      4 - 3 = 1
5       5               Discard         Discard
6       7               6 - 7 = -1      7 - 6 = 1

Cumulative scores       -2              2
```

Since Andrea's score (2) is greater than Maria's score (-2), Andrea wins.
Return the name of the winner ("Andrea", "Maria") or "Tie" if they have equal scores.

Function Description

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

int andrea[n]: Andrea's array of card values.
int maria[n]: Maria's array of card values.
string s: the starting called-out word, "Even" or "Odd"

Return

string: either "Maria", "Andrea", or "Tie"

Constraints

- $2 \le n \le 10^5$
- $1 \le andrea[i], maria[i] \le 10^3$, where $0 \le i < n$

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of elements in *andrea*.
The next *n* lines each contain an integer describing *a[i]*, where $0 \le i < n$.
The next line contains an integer, *n*, denoting the number of elements in *maria*.
The next *n* lines each contain an integer describing *maria[i]*, where $0 \le i < n$.
The next line contains the string *s*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
3       →   andrea[] size n = 3
1       →   andrea = [1, 2, 3]
2
3
3       →   maria[] size n = 3
2       →   maria = [2, 1, 3]
1
3
Even    →   s = 'Even'
```

Sample Output 0

```
Maria
```

Explanation 0

The indices range from *0* through *2*. Since *s* = *'Even'*, the only cards flipped are at indices *0* and *2*.

- When *i* = 0, Andrea gets *andrea[0]* − *maria[0]* = 1 − 2 = -1 point and Maria gets *maria[0]* − *andrea[0]* = 2 − 1 = 1 point.
- When *i* = 2, Andrea gets *andrea[2]* − *maria[2]* = 3 − 3 = 0 points and Maria gets *maria[2]* − *andrea[2]* = 3 − 3 = 0 points.

At the end of play, Andrea's cumulative score is -1, and Maria's is 1.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
--------    --------
3       →   friend_1[] size n = 3
1       →   friend_1 = [1, 2, 3]
2
3
3       →   friend_2[] size n = 3
2       →   friend_2 = [2, 1, 3]
1
3
Odd     →   s = 'Odd'
```

Sample Output 1

```
Andrea
```

Explanation 1

Since *s* = "Odd", the only index flipped is at index *1*.

- When *i* = 1, Andrea gets *andrea[1]* − *maria[1]* = 2 − 1 = 1 point and Maria gets *maria[1]* − *andrea[1]* = 1 − 2 = -1 point.

Andrea ends with 1 point, and Maria with -1.

Question - 86
Validating Binary Strings with RegEx

Write a regular expression to validate a binary string that satisfies these constraints:

- It starts with the character 1.
- The string contains at least two 0's.
- The string contains an even number of 0's.

For example, "1010001" is valid, but "01000" and "1000" are not.

Replace the blank in the code with a regular expression that matches valid strings according to these criteria. Locked code in the editor prints *True* for each correct match and *False* for each incorrect match.

Function Description
Replace the blank in the editor below with a regex expression. If your expression matches, the code will print *True*. Otherwise it will print *False*.

Constraints
- $1 \leq queries \leq 10^3$
- $1 \leq$ string length $\leq 10^3$

▼ Input Format for Custom Testing
The first line contains an integer *queries*, the number of target strings to match.
Each of the next *queries* lines contains a string to match.

▼ Sample Case 0
Sample Input

```
5
1001
11000
0101
1010
1111
```

Sample Output

```
True
False
False
True
False
```

Explanation
We perform the following *queries* = 5 validations:

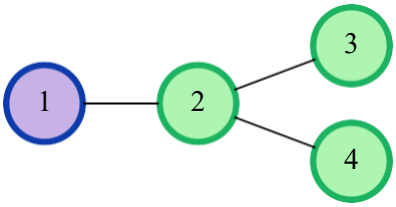
1. "1001" starts with a 1 and contains an even number of 0's, so it is valid.
2. "11000" starts with a 1 but contains an odd number of 0's, so it is not valid.
3. "0101" starts with a 0, so it is not valid.
4. "1010" starts with a 1 and contains an even number of 0's, so it is valid.
5. "1111" starts with a 1 but doesn't contain at least two 0's, so it is not valid.

Question - 87
Delivery Management System

A manufacturing company needs to ship goods to cities connected by bidirectional roads. Determine the order of cities for delivery based on these rules:

- Cities are numbered from 1 to *n*.
- Some cities might be inaccessible due to lack of connecting roads.
- Delivery order is determined first by distance from the manufacturing company.
- If multiple cities are at the same distance, prioritize the city with the smaller number.

For example, say that the number of cities is *cityNodes = 4*, where *cityFrom = [1, 2, 2]*, *cityTo = [2, 3, 4]*, and *company = 1*. The company is located in city 1, and the roads run between cities 1 and 2, cities 2 and 3, and cities 2 and 4, like so:



In this case, the cities are visited based on the following logic:

- The closest city (or cities) is visited first. This is city 2, which is 1 unit from the manufacturing company.
- The next-closest cities are visited next. City 3 and city 4 are both 2 units from the company.
  - In this case, the lower-numbered city is prioritized: visit city 3 first, then city 4.

Therefore, the answer is [2, 3, 4].

Function Description

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

int *cityNodes:* the number of cities
int *cityFrom[n]:* the first city node where there is a bidirectional edge
int *cityTo[n]:* the second city node where there is a bidirectional edge
int *company:* the node where the route starts

Returns

int[]: the cities in the order visited

Constraints

- $2 \le cityNodes \le 10^5$
- $1 \le n \le min(^{(cityNodes \times (cityNodes - 1))}/_{[2]}, 10^5)$
- $1 \le cityFrom[i], cityTo[i], company \le n$
- $cityFrom[i] \ne cityTo[i]$

▼ Input Format For Custom Testing

The first line contains two space-separated integers: *cityNodes,* denoting the number of cities, and *n,* denoting the number of roads.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains two space-separated integers, *cityFrom[i]* and *cityTo[i]*
The next line contains an integer, *company*.

▼ Sample Case 0

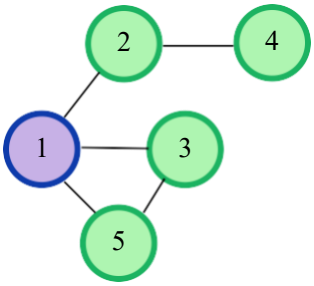
Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
5 5    →    cityNodes = 5, n = 5
1 2    →    cityFrom = 1, cityTo = 2
1 3    →    cityFrom = 1, cityTo = 3
2 4    →    cityFrom = 2, cityTo = 4
3 5    →    cityFrom = 3, cityTo = 5
1 5    →    cityFrom = 1, cityTo = 5
1      →    company = 1
```

Sample Output

```
2
3
5
4
```

Explanation



Cities 2, 3, and 5 are all 1 unit of distance away from the company. These are visited based on priority in ascending order, so [2, 3, 5]. City 4 is 2 units of distance, so it is visited next. The final order is [2, 3, 5, 4].

▼ Sample Case 1

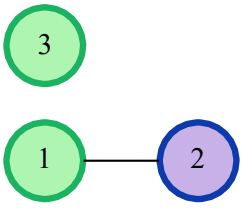
Sample Input For Custom Testing

```
STDIN       Function
-------     -----------
3 1    →    cityNodes = 3, n = 1
1 2    →    cityFrom = 1, cityTo = 2
2      →    company = 2
```

Sample Output

```
1
```

Explanation



City 1 is 1 unit of distance from the manufacturing company. City 3 is not accessible because there are no roads connecting it to the manufacturing company's city. Therefore, the answer is [1].

Question - 88
Multiple Requests at a Time

You are given an input filename. You must read this file, extract timestamps from each line, and create a new file containing all timestamps that occur more than once. All timestamps are from July, 1995.

Naming convention:

- Input file: "filename"
- Output file: "req_*filename*" (replace "*filename*" with the actual name)

For example, if the input file is "hosts_access_log_00.txt", the output file should be "req_hosts_access_log_00.txt".

Each line in the input file contains a log record with the following format:

```
hostname - - [timestamp] "request" response_code bytes
```

The timestamp format is "[DD/mmm/YYYY:HH:MM:SS -0400]", where:

- DD is the day of the month
- mmm is the name of the month
- YYYY is the year
- HH:MM:SS is the time in 24-hour format
- -0400 is the time zone

For example:

```
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
```

Your task is to:

1. Extract the timestamp from each line
2. Identify timestamps that appear in multiple log records
3. Write these timestamps to the output file in the format DD/mmm/YYYY:HH:MM:SS, one per line

The line order in the output file does not matter.

Note: The output file has to be written to the current directory.

Constraints

- The log file contains no more than 2×10^5 records.

▼ Input Format for Custom Testing

The only line contains a string, *filename*.

▼ Sample Case 0

Sample Input

```
hosts_access_log_00.txt
```

Sample Output

Given *filename* = *"hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *req_hosts_access_log_00.txt* containing the following rows:

```
01/Jul/1995:00:00:12
01/Jul/1995:00:00:14
01/Jul/1995:00:00:15
```

Explanation 0

The log file *hosts_access_log_00.txt* contains the following log records:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

The data confirms the following:

1. The timestamp 01/Jul/1995:00:00:12 occurs two times.
2. The timestamp 01/Jul/1995:00:00:14 occurs three times.
3. The timestamp 01/Jul/1995:00:00:15 occurs two times.

Strip the brackets and time zones from the three timestamps occurring more than once and append them to the output file.

Question - 89

Hosts and the Total Number of Requests

In this challenge, write a program to analyze a log file and summarize the results. Given a text file of an http requests log, list the number of requests from each host. **Output should be directed to a file as described in the Program Description below.**

The format of the log file, a text file with a *.txt* extension, follows. Each line contains a single log record with the following columns (in order):

1. The *hostname* of the *host* making the request.
2. This column's values are missing and were replaced by a hyphen.
3. This column's values are missing and were replaced by a hyphen.
4. A timestamp enclosed in square brackets following the format *[DD/mmm/YYYY:HH:MM:SS -0400]*, where *DD* is the day of the month, *mmm* is the name of the month, *YYYY* is the year, *HH:MM:SS* is the time in *24*-hour format, and *-0400* is the time zone.
5. The *request*, enclosed in quotes (e.g., *"GET /images/NASA-logosmall.gif HTTP/1.0"*).
6. The *HTTP response code*.
7. The total number of *bytes* sent in the response.

▼ Example log file entry

Given the following log record:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
```

We can label each column in the record like so:

Hostname	-	-	Timestamp	Request	HTTPResponseCode	Bytes
unicomp6.unicomp.net	-	-	[01/Jul/1995:00:00:0	"GET/shuttle/countdow	200	3985

6-0400] n/HTTP/1.0"

Function Description

Your function must create a unique list of hostnames with their number of requests and **output to a file named *records_filename*** where ***filename*** is replaced with the input *filename*. Each hostname should be followed by a space, the number of requests, and a newline. Order does not matter.

Constraints

- The log file has a maximum of 2×10^5 lines of records.

▼ Input Format

There is one line of input which contains the string *filename* read from STDIN.

▼ Sample Case 0

Sample Input 0

```
hosts_access_log_00.txt
```

Sample Output 0

Given *filename = "hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *records_hosts_access_log_00.txt* which contains the following rows in any order:

```
burger.letters.com 3
d104.aa.net 3
unicomp6.unicomp.net 4
```

Explanation 0

The log file *hosts_access_log_00.txt* contains the following log records:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

When the data is consolidated, it confirms the following:

1. The host *unicomp6.unicomp.net* made *4* requests.
2. The host *burger.letters.com* made *3* requests.
3. The host *d104.aa.net* made *3* requests.

Question - 90

Separate the Files

Consider a text file that contains a list of *C*, *C++*, and *C#* source code file names. The extensions for the file types are as follows:

1. C → *.c*
2. C++ → *.cpp*
3. C# → *.cs*

Given a string, *baseFilename*, that denotes the name of a real text file, **create the following three output files** where *baseFilename* denotes the name of the file received as input and the base of the names for output files:

1. *c_baseFilename*, for storing *C* file names.
2. *cpp_baseFilename*, for storing *C++* file names.
3. *cs_baseFilename*, for storing *C#* file names.

Next, process the list of file names in *baseFilename* in order and, for each source code file name encountered, append its name to the appropriate output file.

For example, the input *baseFilename* is *file_00.txt* and it contains the following data:

```
first.c
first.cpp
first.cs
second.c
```

First, create the three files named *c_file_00.txt*, *cpp_file_00.txt* and *cs_file_00.txt*. Read in each source file name and write them out to their appropriate files. The results are:

- *c_file_00.txt*

  ```
  first.c
  second.c
  ```

- *cpp_file_00.txt*

  ```
  first.cpp
  ```

- *cs_file_00.txt*

  ```
  first.cs
  ```

The file names must maintain the order in which they were listed in *file_00.txt*.

Note: There may not be a file of every type included in the input. In that case, the output file for that type should still be created and it should be empty.

Function Description

Complete the code in the editor below. It must create three files and save the appropriate source file names to each one while maintaining order.

The program has the following parameter(s):

baseFilename: the name of a text file

Constraints

- *baseFilename* has a maximum of 10^5 lines.

▼ Input Format for Custom Testing

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

The only line contains the name of a text file to read, *baseFilename*.

▼ Sample Case 0

Sample Input 0

```
names_list_00.txt
```

Sample Output 0

Create the following three output files:

- *c_names_list_00.txt*:

```
code1.c
code3.c
code4.c
```

- *cpp_names_list_00.txt*:

```
code1.cpp
code5.cpp
```

- *cs_names_list_00.txt*:

```
code1.cs
code2.cs
```

Explanation 0

The content of the text file *names_list_00.txt* is:

```
code1.c
code1.cpp
code1.cs
code2.cs
code3.c
code4.c
code5.cpp
```

Observe that each source code file name in the input file was put into the output file corresponding to its language, and the order of the source code file names is maintained.

Question - 91

Minimum Moves

Given two arrays of integers, *arr1* and *arr2*, find the minimum number of moves needed to match *arr1* with *arr2*. One move is defined as incrementing or decrementing one digit of any element in an array. No reordering of digits is allowed.

Example

arr1 = [123, 543]

arr2 = [321, 279]

To match *arr1[0]* = 123 with *arr2[0]* = 321:

- Increment 1 twice to get 3 (2 moves)
- Decrement 3 twice to get 1 (2 moves)
- Total: 4 moves

To match *arr1[1]* = 543 with *arr2[1]* = 279:

- Decrement 5 three times to get 2 (3 moves)
- Increment 4 three times to get 7 (3 moves)
- Increment 3 six times to get 9 (6 moves)
- Total: 12 moves

The total number of moves required is 4 + 12 = 16.

Function Description

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

int arr1[n]: array to modify

int arr2[n]: array to match

Returns

int: the minimum number of moves to match *arr1* with *arr2*

Constraints

- $1 \le n \le 10^5$
- $1 \le arr1[i], arr2[i] \le 10^9$
- The lengths of *arr1* and *arr2* are equal, $|arr1| = |arr2|$.
- The elements *arr1[i]* and *arr2[i]* have an equal number of digits.

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr1.*
The next *n* lines each contain an element *arr1[i]* where $0 \le i < n$.
The next line contains an integer *n*, the size of the array *arr2*.
The next *n* lines each contain an element *arr2[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
2       →   n = 2
1234    →   arr1 = [1234, 4321]
4321
2       →   n = 2
2345    →   arr2 = [2345, 3214]
3214
```

Sample Output

```
10
```

Explanation

- Match *arr1[0]*=1234 with *arr2[0]*=2345.
  - Increment 1 once to get 2 (1 move)
  - Increment 2 once to get 3 (1 move)
  - Increment 3 once to get 4 (1 move)
  - Increment 4 once to get 5 (1 move).
  - 4 moves are needed to match 1234 with 2345.
- Match *arr1[1]*=4321 with *arr2[1]*=3214.
  - Decrement 4 once to get 3 (1 move)
  - Decrement 3 once to get 2 (1 move)
  - Decrement 2 once to get 1 (1 move)
  - Increment 1 three times to get 4 (3 moves)
  - 6 moves are needed to match 4321 with 3214.
- 6+4=10 total moves are needed to match the arrays *arr1* and *arr2*.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
1       →   n = 1
2468    →   arr1 = [2468]
1       →   n = 1
```

```
8642  →   arr2 = [8642]
```

Sample Output

```
16
```

Explanation

- Match *arr1[0]*=2468 with *arr2[0]*=8642.
  - Increment 2 six times to get 8 (6 moves)
  - Increment 4 twice to get 6 (2 moves)
  - Decrement 6 twice to get 4 (2 moves)
  - Decrement 8 six times to get 2 (6 moves).

Question - 92
Twin Strings

Two strings are called twins if they can be made equivalent by performing some number of operations on one or both strings. The possible operations are:

- SwapEven: Swap a character at an even-numbered index with a character at another even-numbered index
- SwapOdd: Swap a character at an odd-numbered index with a character at another odd-numbered index

Given two string arrays, compare each pair of strings at the same index and determine if they are twins or not.

Example

firstString = ["abcd", "abcd"]
secondString = ["cbad", "adbc"]

Comparing *firstString[0]* and *secondString[0]*:

- We can apply one *SwapEven* operation to swap 'a' and 'c' in "abcd" to get "cbad".
- Therefore, these strings are twins, and the answer is "Yes".

Comparing *firstString[1]* and *secondString[1]*:

- No combination of *SwapOdd* or *SwapEven* operations can make "abcd" equivalent to "adbc".
- Therefore, these strings are not twins, and the answer is "No".

The result is ["Yes", "No"].

Function Description

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

string firstString[n]: first array of strings
string secondString[n]: second array of strings

Returns

string[n] : array of strings containing the string "Yes" if *firstString[i]* and *secondString[i]* are twins or the string "No" otherwise.

Constraints

- $1 \le n \le 10^3$
- $1 \le |firstString[i]|, |secondString[i]| \le 100$ where $|s|$ means "the length of s"
- *firstString[i]* and *secondString[i]* may not have equal lengths.
- Strings *firstString[i]* and *secondString[i]* contain lowercase letters only, in the range ascii[a-z].

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *the array firstString*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string describing *firstString[i]*.
The first line contains an integer, *n*, denoting the number of elements in the array *secondString*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string describing *secondString[i]*

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
3       →   firstString[] size n = 3
cdab    →   firstString = ["cdab","dcba","abcd"]
dcba
abcd
3       →   secondString[] size n = 3
abcd    →   secondString = ["abcd","abcd","abcdcd"]
abcd
abcdcd
```

Sample Output 0

```
Yes
No
No
```

Explanation 0

Given the two string arrays *firstString*= ["cdab", "dcba", "abcd"] and *secondString* = ["abcd", "abcd", "abcdcd"], we process each element as follows:

- Compare the two strings f*irstString*[0]="cdab" and *secondString*[0]="abcd". One SwapEven and one SwapOdd of "cdab" allow us to swap the character "c" with "a" and "d" with"b" and make it equivalent to "abcd" ("**cda**b" → "a**dcb**" → "abcd"). f*irstString*[0] and *secondString*[0] are twins and result[0]="Yes"
- Compare the two strings f*irstString*1]="dcba" and *secondString*[1]="abcd". No SwapOdd or SwapEven operations can make the two strings abcd and adbc equivalent. f*irstString*[1] and *secondString*[1] are not twins and result[1]="No"
- Compare the two strings f*irstString*[2]="abcd" and *secondString*[2]="abcdcd". The two strings have different lengths and can never be equivalent. f*irstString*[2] and *secondString*[2] are not twins and result[2]="No"
- result = ["Yes","No","No"]

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-----       --------
2       →   firstString[] size n = 2
abbc    →   firstString = ["abbc","abbdd"]
abbdd
2       →   secondString[] size n = 2
abbc    →   secondString = ["abbc","ddbba"]
ddbba
```

Sample Output 1

```
Yes
Yes
```

Explanation 1

Given the two string arrays f*irstString* = ["abbc","abbdd"] and *secondString* = ["abbc","ddbba"], we process each element as follows:

- Compare the two strings f*irstString*[0]="abbc" and *secondString*[0]="abbc". f*irstString*0] and *secondString*[0] are equivalent and twins without any operation and result[0]="Yes"
- Compare the two strings f*irstString*[1]="abbdd" and *secondString*[1]="ddbba". One SwapEven and one SwapOdd of "abbdd" allow us to swap the character "a" with "d" and "d" with"b" and make it equivalent to "ddbba" ("**a**bbd**d**" → "d**bbd**a" → "d**dbb**a"). f*irstString*[1] and *secondString*[1] are twins and result[1]="Yes"
- result= ["Yes","Yes"]

Question - 93
Balanced Or Not?

Determine if each string can be balanced within a given number of replacements.

A string consists solely of the characters '<' and '>'. The string is considered balanced if every '<' appears before a corresponding '>' character. The characters do not have to be adjacent, and each '<' and '>' forms a unique pair that cannot be part of another pair.

To balance a string, any '>' character can be replaced with the sequence '<>'.

Given an array of expressions and a corresponding array of maximum replacements, determine if each string can be balanced. Return an array where the i^{th} value is 1 if *expressions[i]* can be balanced using up to *maxReplacements[i]* replacements, or 0 otherwise.

Example

expressions = ['<<>>', '<>', '<><>', '>>', '<<>', '><><']

maxReplacements = [0, 1, 2, 2, 2, 2]

Process a series of *expressions* and their corresponding *maxReplacements*. Each of the first three expressions is balanced. The string *expressions[3] = '>>'* can be balanced in two moves by replacing each `>` with a `<>` to make `<><>`. Neither of the last two strings can ever be balanced. The answer to return is [1, 1, 1, 1, 0, 0].

Function Description

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

string expressions[n]: the strings to check

int maxReplacements[n]: the maximum number of replacements available for each *expressions[i]*

Returns:

int[n]: each *element[i]* contains a *1* if *expressions[i]* is balanced or a *0* if it is not

Constraints

- $1 \le n \le 10^2$
- $1 \le$ *length of expressions[i]* $\le 10^5$
- $0 \le$ *maxReplacements[i]* $\le 10^5$

▼ 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 *expressions*.

The next *n* lines each contain a string, *expressions[i]*.

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

The next *n* lines each contain an integer, *maxReplacements[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
2       →   expressions[] size n = 2
<>>>    →   expressions = ['<>>>', '<>>>>']
<>>>>
2       →   maxReplacements[] size n = 2
2       →   maxReplacements = [2, 2]
2
```

Sample Output

```
1
0
```

Explanation

0. For the string `<>>>` with *maxReplacements[0]* = *2*, it becomes balanced after two replacements: `<>>>` → `<><>>` → `<><><>`. The string was converted in ≤ *maxReplacements[0]* replacements. Store a *1* in index *0* of the return array.
1. For the string `<>>>>` with *maxReplacements[1]* = *2*, it becomes balanced after three replacements: `<>>>>` → `<><>>>` → `<><><>>` → `<><><><>`. There were not enough replacements available, so store a *0* in index *1* of the return array.

Return the array *[1, 0]* as the answer.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
2       →   expressions[] size n = 2
<>      →   expressions = ['<>', '<>><']
<>><
2       →   maxReplacements[] size n = 2
1       →   maxReplacements = [1, 0]
0
```

Sample Output

```
1
0
```

Explanation

0. For the string `<>` with *maxReplacements[0]* = *1*, it is already balanced and needs no replacements. Store a *1* in index *0* of the return array.
1. For the string `<>><` with *maxReplacements[1]* = *0*, the string is not balanced. It is impossible to balance the string because it ends in `<` and because there are *0* replacements available. Store a *0* in index *1* of the return array.

Return the array *[1, 0]* as the answer.

▼ Sample Case 2

Sample Input

```
STDIN         Function
-----         --------
1         →   expressions[] size n = 1
<<<><><>  →   expressions = ['<<<><><>']
1         →   maxReplacements[] size n = 1
2         →   maxReplacements = [2]
```

Sample Output

```
0
```

Explanation

0. For string `<<<><><>` with *maxReplacements[0]* = *2*, the string is not balanced. It is impossible to balance because there are more `<` than `>`. Store a *0* in index *0* of the return array.

Return the array *[0]* as the answer.

Question - 94
Count Between

Given:

- An array of integers *arr*

- Two arrays *low* and *high* representing the lower and upper bounds of each range query

For each range query, count how many elements in *arr* have values between *low[i]* and *high[i]* inclusive.

Example

arr = [1, 2, 2, 3, 4]

low = [0, 2]

high = [2, 4]

Query 1: *low* = 0, *high* = 2

- Element in range [0, 2]: [1, 2, 2]
- Count: 3

Query 2: *low* = 2, *high* = 4

- Elements in range [2, 4]: [2, 2, 3, 4]
- Count: 4

Return [3, 4].

Function Description

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

int arr[n]: the array to analyze

int low[q]: the lower range limits

int high[q]: the higher range limits

Example

int[q]: the answers to the queries

Constraints

- $1 \le n \le 10^5$
- $1 \le arr[j] \le 10^9$
- $1 \le q \le 10^5$
- $1 \le low[i] \le high[i] \le 10^9$

▼ Input Format For Custom Testing

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

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

The next line contains an integer, *q*, that denotes the number of elements in *low[q]*.

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

The next line repeats the integer, *q*, the number of elements in *high[q]*.

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

▼ Sample Case 0

Sample Input For Custom Testing

Sample Input 0

```
STDIN       Function
-----       --------
5     →     arr[] size n = 5
1     →     arr = [1, 3, 5, 6, 8]
3
5
6
8
1     →     low[] size q = 1
2     →     low = [2]
1     →     high[] size q = 1
6     →     high = [6]
```

Sample Output 0

```
3
```

Explanation 0

Query 0: There are 3 elements in the inclusive range [2, 6]: [3, 5, 6], so store 3 in index 0 of the return array.

▼ Sample Case 1

Sample Input For Custom Testing

Sample Input 1

```
STDIN       Function
-----       --------
3     →     arr[] size n = 3
4     →     arr = [4, 8, 7]
8
7
2     →     low[] size q = 2
2     →     low = [2, 4]
4
2     →     high[] size q = 2
8     →     high = [8, 4]
4
```

Sample Output 1

```
3
1
```

Explanation 1

Query 0: There are 3 elements in the inclusive range [2, 8]: [4, 7, 8] so store 3 in index 0 of the return array.
Query 1: There is 1 element in the inclusive range [4, 4]: [4] so store 1 in index 1 of the return array.

Question - 95
Arrange the Words

A sentence is a string of words separated by spaces, starting with a capital letter, followed by lowercase letters and spaces, and ending with a period. This matches the regular expression *^[A-Z][a-z]*.$*. Rearrange the words in the sentence according to the following rules:

- Words are ordered by length in ascending order.
- Words of the same length must retain their original order.
- The rearranged sentence must conform to the regular expression *^[A-Z][a-z]*.$*.

Note: *^[A-Z][a-z]*.$* means that it begins with a capital letter, is followed by lowercase letters and spaces, then ends in a period.

Example

sentence = 'Cats and hats.'

Order the sentence by the words' lengths, and keep the original order for the words with the same length.

- Length 3: {and}
- Length 4: {Cats, hats}

Reassemble the sequence of words so that the first letter is uppercase, the intermediate letters are lowercase, and the last one is a period.
The result is 'And cats hats.'

Function Description

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

string sentence: a well-formed sentence string

Returns

string: a formatted, rearranged sentence string

Constraints

- $1 \leq$ length of *sentence* $< 10^5$

▼ Input Format for Custom Testing

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

▼ Sample Case 0

Sample Input 0

```
STDIN                                       Function
-----                                       --------
The lines are printed in reverse order.  →  sentence = 'The lines are printed in reverse order.'
```

Sample Output 0

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

Explanation 0

Sort the words by length. Keep the original order for the words with the same length.

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

Reassemble the sequence of words, make the first letter uppercase, the intermediate letters lowercase, and the last character a period.

▼ Sample Case 1

Sample Input 1

```
STDIN            Function
-----            --------
Here i come.  →  sentence = 'Here i come.'
```

Sample Output 1

```
I here come.
```

Explanation 1

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

▼ Sample Case 2

Sample Input 2

```
STDIN               Function
-----               --------
I love to code. →   sentence = 'I love to code.'
```

Sample Output 2

```
I to love code.
```

Explanation 2

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

Question - 96
The Perfect Team

The School of Languages and Science offers five subjects: Physics, Chemistry, Math, Botany, and Zoology. Each student is proficient in one subject. The students' skills are represented by a string of characters, with each character being one of the following: p, c, m, b, or z, indicating their respective skills.

Given a list of students' skills, calculate the total number of possible teams that meet the following conditions:

- A team must consist of exactly five students.
- Each student on the team must be proficient in a different subject.
- A student can only be part of one team.

Example 1

skills = pcmbzpcmbz

There are 2 possible teams that can be formed at one time: *skills*[0-4] = pcmbz and *skills*[5-9] = pcmbz, for example.

Example 2

skills = mppzbmbpzcbmpbmczcz

The sorted string is *bbbbcccmmmmpppzzzz*. All of the skills are represented, but only 3 students are skilled in Chemistry. Only 3 teams can be created.

Function Description

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

string skills: a string of length *n*, where each position represents the skill of a student

Returns

int: the number of teams that can be formed

Constraints

- $5 \le n \le 5 \times 10^5$
- *skills[i] are in the set {p,c,m,b,z}*

▼ Input Format for Custom Testing

The only line contains a string, *skills*.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-------   -----------
pcmbz →   string skills = "pcmbz"
```

Sample Output 0

```
1
```

Explanation 0

Each of the necessary skills appears once in the string. Only 1 team can be formed.

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-------   -----------
pcmpp →   string skills = "pcmpp"
```

Sample Output 1

```
0
```

Explanation 1

There are no students skilled in Zoology or Botany. No team can be formed.

▼ Sample Case 2

Sample Input 2

```
STDIN           Function
-----           --------
pcmpcmbbbzz →   string skills = "pcmpcmbbbzz"
```

Sample Output 2

```
2
```

Explanation 2

The sorted list of characters is *bbbccmmppzz*. There are 3 students skilled in Biology, but only 2 in each of the other subjects. Only 2 teams can be formed.

Question - 97

Longest Even Length Word

Given a string consisting of words separated by spaces, with each word containing only English letters, find and return the first word that has an even length and is the longest among all even-length words in the sentence. If there are no even-length words, 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 with 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

- $1 \le length\ of\ sentence \le 10^5$
- The *sentence* string consists of spaces and letters in the range ascii[a-z, A-Z,] only.

▼ Input Format for Custom Testing

Input is a single line of space-separated strings, *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*).

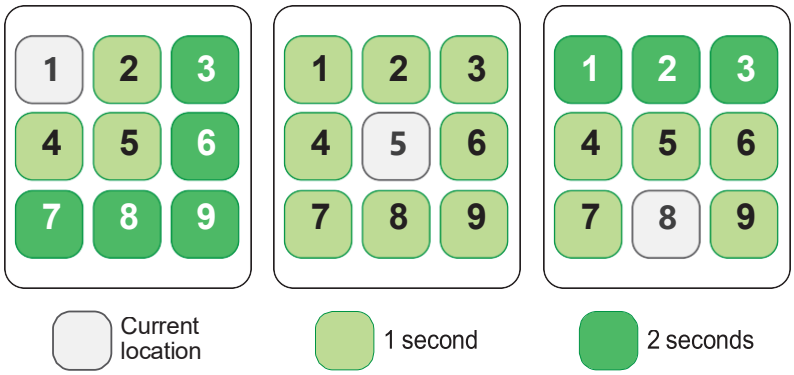
Question - 98
Keyboard

Calculate the minimum time needed to type a string of digits on a numeric keypad where the key positions are mixed up.
Rules:

- It takes 0 seconds to move to the first key that you press.
- It takes 0 seconds to press the key where your finger is currently located.
- Moving to an adjacent key (including diagonals) takes 1 second.
- Moving to a non-adjacent key requires a series of moves to adjacent keys.

Your task is to find the most efficient path to type the given string and return the minimum time required.

Example



This diagram depicts the minimum amount of time it takes to move from the current location to all other locations on the keypad.

Function Description

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

string s: the string to type

string keypad: a string of 9 digits where each group of 3 digits represents a row on the keypad, in order

Returns:

int : integer denoting the minimum amount of time it takes to type the string *s*

Constraints

- $1 \leq$ length of $s \leq 10^5$
- length of *keypad* = 9
- *keypad[i]* is in the range [1-9]

▼ Input Format for Custom Testing

The first line contains a string *s*.

The next line contains a string *keypad*.

▼ Sample Case 0

Sample Input 0

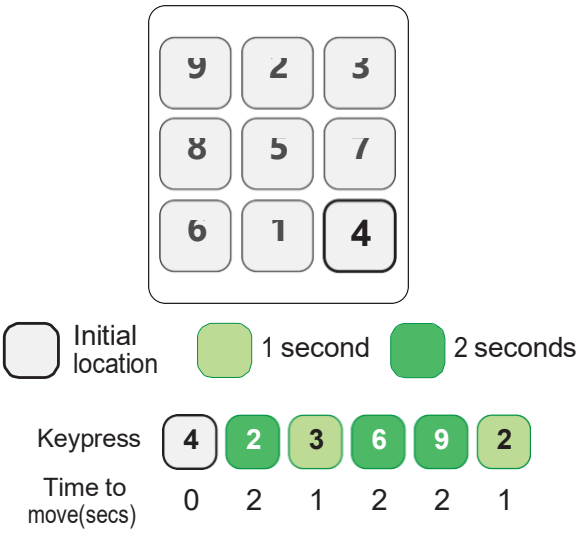
```
STDIN           Function Parameters
-------         -------------------------
423692     →    string s = "423692"
923857614  →    string keypad = "923857614"
```

Sample Output 0

```
8
```

Explanation 0

The keypad looks like this:



Time to type 423692 = 0+2+1+2+2+1 = 8 seconds

Calculate the time it takes to type *s* = "423692" as follows:

- 4: Start here, so it takes 0 seconds.
- 2: It takes 2 seconds to move from 4 → 2
- 3: It takes 1 second to move from 2 → 3
- 6: It takes 2 seconds to move from 3 → 6
- 9: It takes 2 seconds to move from 6 → 9
- 2: It takes 1 second to move from 9 → 2

The total time is 2 + 1 + 2 + 2 + 1 = 8.

▼ Sample Case 1

Sample Input 1

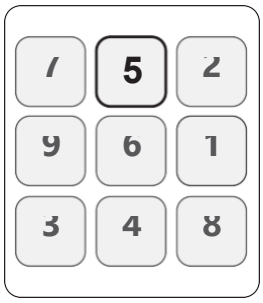
```
STDIN          Function Parameters
-------        --------------------------
5111       →   string s = "5111"
752961348  →   string keypad = "752961348"
```

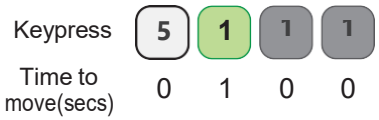
Sample Output 1

```
1
```

Explanation 1

The keypad looks like this:





Time to type 5111 = 0+1+0+0 = 1 second

Calculate the time it takes to type *s* = "5111" as follows:

- 5: Start here, so it takes 0 seconds, and *totalTime* = 0.
- 1: It takes 1 second to move from 5 → 1
- 1: It takes 0 seconds to move from 1 → 1
- 1: It takes 0 seconds to move from 1 → 1

▼ Sample Case 2

Sample Input 2

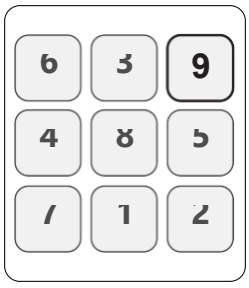
```
STDIN          Function Parameters
-------        -------------------------
91566165  →    string s = "91566165"
639485712 →    string keypad = "639485712"
```

Sample Output 2

```
11
```

Explanation 2

The keypad looks like this:



Time to type 91566165 = 0+2+1+2+0+2+2+2 = 11 seconds

Calculate the time it takes to type *s* = "91566165" as follows:

- 9: Start here, so it takes 0 seconds.

- 1: It takes 2 seconds to move from 9 → 1
- 5: It takes 1 second to move from 1 → 5
- 6: It takes 2 seconds to move from 5 → 6
- 6: It takes 0 seconds to move from 6 → 6
- 1: It takes 2 seconds to move from 6 → 1
- 6: It takes 2 seconds to move from 1 → 6
- 5: It takes 2 seconds to move from 6 → 5,

Question - 99
Find the Factor

Find all factors of a given number *n* and return the p^{th} smallest factor using 1-based indexing. A factor is a positive integer that divides evenly into *n*.

If there is no p^{th} element in the list of factors, return 0.

Example
n = 20
p = 3

The factors of 20 in ascending order are {1, 2, 4, 5, 10, 20}. The 3rd factor (using 1-based indexing) is 4.
If *p* were greater than 6, the answer would be 0 since there are only 6 factors.

Function Description
Complete the function *pthFactor* in the editor with the following parameter(s):
long int n: the integer to factor
long int p: the index of the factor to return

Returns:
long int: the value of the p^{th} integer factor of *n,* or, if there is no factor at that index, then return 0

Constraints
- $1 \le n \le 10^{15}$
- $1 \le p \le 10^{9}$

▼ Input Format for Custom Testing

The first line contains an integer *n*.
The second line contains an integer *p*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
----------  --------------------
10      →   n = 10
3       →   p = 3
```

Sample Output 0

```
5
```

Explanation 0
Factoring *n* = 10 results in {1, 2, 5, 10}. Return the 3rd factor.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
----------  ----------
10      →   n = 10
5       →   p = 5
```

Sample Output 1

```
0
```

Explanation 1

Factoring $n = 10$ results in {1, 2, 5, 10}. There are only 4 factors, and $p = 5$.

▼ Sample Case 2

Sample Input 2

```
STDIN       Function
--------    ------------
1       →   n = 1
1       →   p = 1
```

Sample Output 2

```
1
```

Explanation 2

Factoring $n = 1$ results in {1}. The 1^{st} factor is returned.

Question - 100

ASCII-Encoded Strings

Decode a numeric string using ASCII values by following these steps:

1. Reverse the given string of digits.
2. Successively pick valid values from the string and convert them to their ASCII equivalents.
3. Some values will have two digits, others three.

Use these ASCII value ranges for decoding:

- A through Z: 65-90
- a through z: 97-122
- Space character: 32

Example

The string "HackerRank" converts to ASCII values: H:72, a:97, c:99, k:107, e:101, r:114, R:82, a:97, n:110, k:107.

This creates the ASCII string: 72979910710111432829711010 7.

After reversing: 701011792841110170199792 7.

To decode, reverse back and convert valid ASCII values to characters.

Function Description

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

string encoded: an encoded string

Returns

string: the original decoded string

Constraints

- $1 \leq$ length of *encoded* $\leq 10^5$
- *encoded[i]* is an ascii character in the range [A-Za-z] or a space character.

▼ Input Format for Custom Testing

The only line contains an encoded string *decode*.

▼ Sample Case 0

Sample Input

```
STDIN                                                    Function
-----                                                    --------
2351101150178235111217991180156234016117114148    →    encoded = "2351101150178235111217991180156234016117114148"
```

Sample Output

```
Truth Always Wins
```

Explanation

Reverse *encoded* to get "8411411711610432651081199712111532871051101153 2". Then replace each ASCII value with its corresponding character:

84	114	117	116	104	32	65	108	119	97	121	115	32	87	105	110	115	32
T	r	u	t	h		A	l	w	a	y	s		W	i	n	s	

▼ Sample Case 1

Sample Input

```
STDIN                                    Function
-----                                    --------
2312179862310199501872379231018117927  →   encoded = "2312179862310199501872379231018117927"
```

Sample Output

```
Have a Nice Day
```

Explanation

Reverse *encoded* to get "7297118101329732781059910132689712132". Then replace each ASCII value with its corresponding character:

72	97	118	101	32	97	32	78	105	99	101	32	68	97	121	32
H	a	v	e		a		N	i	c	e		D	a	y	

.

▼ Sample Case 2

Sample Input

```
STDIN                                                                                Function
-----                                                                                --------
12199501801111082361151110166231014016112351150123121611511101011111 27  →  encoded  =
"121995018011110823611511101662310140161123511501231216115111010111112 7"
```

Sample Output

```
Honesty is the Best Policy
```

Explanation

Reverse *encoded* to get "7211111010111511612132105115321161041013266101115116328011110810599121". Then replace each ASCII value with its corresponding character:

72	111	110	101	115	116	121	32	105	115	32	116	104	101	32	66	101	115	116	32	80	111	108	105	99	121
H	o	n	e	s	t	y		i	s		t	h	e		B	e	s	t		P	o	l	i	c	y

Question - 101
No Pairs Allowed

For each word in a list, determine the minimum number of character replacements needed so that no two adjacent characters are the same.
If any two adjacent characters in a string are equal, one of them must be changed. Calculate the minimum number of substitutions required for each word.

Example
words = ['add', 'boook', 'break']

- 'add': change one 'd' (1 change)
- 'boook': change the middle 'o' (1 change)
- 'break': no changes necessary (0 changes)

The return array is [1, 1, 0].

Function Description
Complete the function *minimalOperations* in the editor with the following parameter(s):

string words[n]: an array of strings

Returns
int[n]: each element *i* is the minimum substitutions for *words[i]*

Constraints
- $1 \le n \le 100$
- 2 ≤ length of *words[i]* ≤ 10^5
- Each character of *words[i]* is in the range ascii[a-z].

▼ Input Format for Custom Testing
The first line contains an integer *n*, the size of the array *words*.
Each of the next *n* lines contains a string *words[i]*.

▼ Sample Case 0
Sample Input 0
```
STDIN          Function Parameters
-------        --------------------------
5          →   words[] Size = 5
ab         →   words[] = [ 'ab','aab','abb', 'abab','abaaaba' ]
aab
abb
abab
abaaaba
```

Sample Output 0
```
0
1
1
0
1
```

Explanation 0
- *words[0]* = 'ab' is already acceptable, so 0 replacements are needed.
- *words[1]* = 'aab' Replace an 'a' with an appropriate character so 1 replacement.
- *words[2]* = 'abb' is not acceptable. Replace a 'b' with an appropriate character, again 1 replacement.

- *words[3]* = 'abab' is already acceptable so 0 replacements are needed.
- *words[4]* = 'abaaaba' is not acceptable. Replace the middle 'a' in 'aaa', 1 replacement.

The return array is [0, 1, 1, 0, 1].

Question - 102
Spreadsheet Notation Conversion

A spreadsheet application allows data to be organized and manipulated within rows and columns of a table. Each individual rectangle within the table is referred to as a cell. The following conventions apply to spreadsheets:

1. Rows are numbered sequentially from top to bottom, starting at 1 and ending at 10^9.
2. Columns are labeled sequentially from left to right, beginning with A and ending with ZZ. After Z, the sequence continues with AA, AB, and so on, resulting in a total of 702 columns per row. All column labels are capital letters.
3. In spreadsheet notation, a cell at the intersection of row R and column C is denoted as RC. For example, the cell at row 7 and column AH is written as 7AH.

Cell numbers are long integers assigned sequentially to each cell from left to right and top to bottom.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
2	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729
3	1405	1406	1407	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423	1424	1425	1426	1427	1428	1429	1430	1431

A portion of a spreadsheet where each cell contains its corresponding *cell number*.

Function Description
Complete the function *getSpreadsheetNotation* in the editor with the following parameter(s):

long int n: cell number

Returns:
string: a string representation of a cell number in spreadsheet notation

Constraints
- $1 \le n \le 10^{12}$

▼ Input Format For Custom Testing
Input from stdin will be processed as follows and passed to the function.

The line contains a long integer *n*.

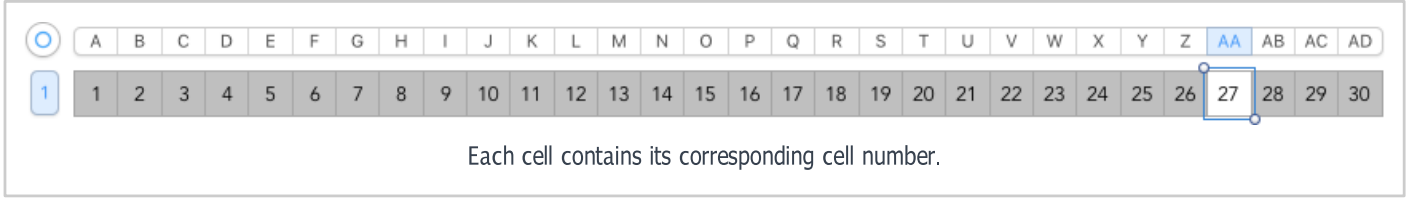
▼ Sample Case 0
Sample Input 0

```
STDIN       Function Parameters
-------     --------------------------
27      →   n = 27
```

Sample Output 0

```
1AA
```

Explanation 0



Cell *n* = *27* is located at the intersection of row *1* and column *AA*.

▼ Sample Case 1

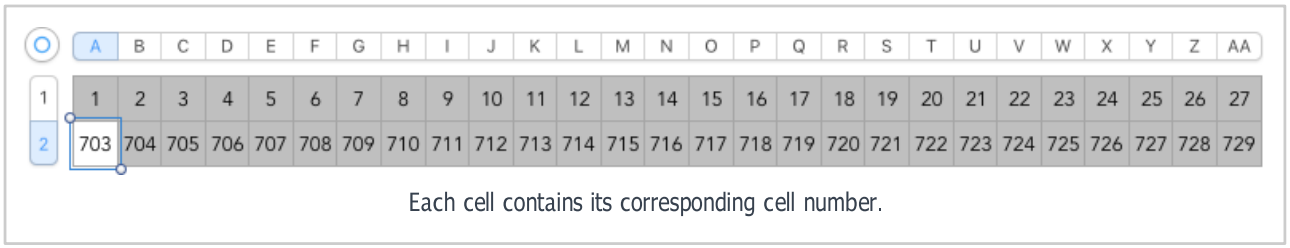
Sample Input 1

```
STDIN       Function Parameters
-------     --------------------------
703     →   n = 703
```

Sample Output 1

```
2A
```

Explanation 1



Cell *n* = *703* is located at the intersection of row *2* and column *A*.

Question - 103
Delete Odd

Given a linked list of integers, return a similar linked list with all elements having odd values removed.

For example, your linked list values are *1* → *2* → *3* → *4*. Your new linked list should be *2* → *4*.

Function Description
Complete the function *deleteOdd* in the editor below. The function must return a reference to the head LinkedListNode of your linked list which has the odd values removed.

deleteOdd has the following parameter(s):
listHead: a reference to the head node of a linked list

Constraints
- $1 \le n \le 10^5$
- $1 \le$ *Linked list node values* $\le 10^5$

▼ 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 number of nodes in the linked list *list*.
Each of the next *n* lines contains an integer *list[i]*, the value associated with the i^{th} node.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       -----------
5       →   list length n = 5
2       →   list = 2 → 1 → 3 → 4 → 6
1
3
4
6
```

Sample Output 0

```
2
4
6
```

Explanation 0

After removing the nodes having odd values, *list = 2 → 4 → 6*.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-----       -----------
5       →   list length n = 5
1       →   list = 1 → 3 → 2 → 7 → 10
3
2
7
10
```

Sample Output 1

```
2
10
```

Explanation 1

After removing the nodes having odd values, *list = 2 → 10*.

Question - 104
Arranging Coins

Using a given number of coins, construct a staircase where each row has one less coin than the row below it. Identify how many complete rows can be formed in the staircase.

Example
coins = [6]

¤
¤ ¤

¤ ¤ ¤

With *6* coins, a *3* row staircase can be built.

Function Description

Complete the function *arrangeCoins* in the editor below.

arrangeCoins has the following parameter(s):

int coins[n]: an array of long integers each representing a number of coins available

Returns

None

Output

For each *coins[i]*, print an integer that denotes the maximum number of complete rows that can be created. Each answer must be on a separate line.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq coins[i] \leq 10^{15}$

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

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-----     -----------
4     →   coins[] size n = 4
2     →   coins = [2, 5, 8, 3]
5
8
3
```

Sample Output 0

```
1
2
3
2
```

Explanation 0

1. *coins[0] = 2*
   The coins can form the following rows:

   ```
   ¤
   ¤
   ```

   Because the 2^{nd} row is incomplete, print *1* on a new line.

2. *coins[1] = 5*
   The coins can form the following rows:

```
¤
¤ ¤
¤ ¤
```

Because the 3^{rd} row is incomplete, print *2* on a new line.

3. *coins[2] = 8*
   The coins can form the following rows:

```
¤
¤ ¤
¤ ¤ ¤
¤ ¤
```

Because the 4^{th} row is incomplete, print *3* on a new line.

4. *coins[3] = 3*
   The coins can form the following rows:

```
¤
¤ ¤
```

Because the 2^{nd} row is complete, print *2* on a new line.

Question - 105
Delete Even

You are given a linked list containing integer values. Your task is to remove all nodes that contain even integer values and return the modified linked list.

Example
n = 3 nodes
listHead = 1 → 4 → 7

The linked list containing only nodes with odd integers is 1 → 7.

Function Description
Complete the function *deleteEven* in the editor with the following parameter(s):
list listHead: a reference to the head of the input linked list

Returns
list: a reference to the head of the resulting linked list.

Constraints
- $1 \le n \le 10^5$
- $1 \le node->data \le 10^5$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the number of nodes in the list.
Each of the next *n* lines contains an integer *node->data*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-------     -----------
5       →   n = 5
```

```
1       →  listHead = 1 → 2 → 3 → 4 → 6
2
3
4
6
```

Sample Output 0

```
1
3
```

The resulting linked list contains only nodes having odd integers: 1 → 3.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
--------    -----------
10      →   n = 10
2       →   listHead = 2 → 4 → 5 → 6 → 8 → 10 → 13 → 23 → 24 → 1
4
5
6
8
10
13
23
24
1
```

Sample Output 1

```
5
13
23
1
```

Explanation

After removing the nodes having even values, the list is 5 → 13 → 23 → 1.

Question - 106
Price Check

There is a shop with old-fashioned cash registers where prices are entered manually, leading to potential errors. Given a list of items and their correct prices, compare them to the prices entered at the time of sale and determine the number of errors.

Example

products = ['eggs', 'milk', 'cheese']
productPrices = [2.89, 3.29, 5.79]
productSold = ['eggs', 'eggs', 'cheese', 'milk']
soldPrice = [2.89, 2.99, 5.97, 3.29].

```
            Price
Product     Actual  Expected  Error
```

```
eggs       2.89    2.89
eggs       2.99    2.89       1
cheese     5.97    5.79       1
milk       3.29    3.29
```

The second sale of eggs has the wrong price, as does the sale of cheese. There are *2* errors in pricing.

Function Description

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

string *products[n]:* each *products[i]* is the name of an item for sale
float *productPrices[n]:* each *productPrices[i]* is the price of *products[i]*
string *productSold[m]:* each *productSold[j]* is the name of a product sold
float *soldPrice[m]:* each *soldPrice[j]* contains the sale price recorded for *productSold[j]*.

Returns

int: the number of sale prices that were entered incorrectly

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le n$
- $1.00 \le productPrices[i], soldPrice[j] \le 100000.00$, where $0 \le i < n$, and $0 \le j < m$

▼ Input Format for Custom Testing

The first line contains an integer *n,* the size of the *products* array.
The next *n* lines each contain an element *products[i]*.
The next line contains an integer *n,* the size of the *productPrices* array.
The next *n* lines each contain an element *productPrices[i]*.
The next line contains an integer *m*, the size of the *productSold* array.
The next *m* lines each contain an element, *productSold[j]*.
The next line contains an integer, *m*, the size of the *soldPrice* array.
The next *m* lines each contain an element *soldPrice[j]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
4        →    products[] size n = 4
rice     →    products=['rice', 'sugar', 'wheat', 'cheese']
sugar
wheat
cheese
4        →  productPrices[] size n = 4
16.89    →  productPrices=[16.89, 56.92, 20.89, 345.99]
56.92
20.89
345.99
2        →  productSold[] size m = 2
rice     →  productSold =['rice', 'cheese']
cheese
2        →  soldPrice[] size m = 2
18.99    →  soldPrice =[18.99, 400.89]
400.89
```

Sample Output 0

```
2
```

Explanation 0

```
          Price
Product   Actual  Expected  Error
rice       18.99   16.89    1
cheese    400.89  345.99    1
```

The sales of *rice* and *cheese* were at the wrong prices. So, the number of sale prices that were entered incorrectly is 2.

▼ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
3          →   n = 3 .The size of the products array
chocolate  →   products=[chocolate, cheese, tomato]
cheese
tomato
3          →   n = 3 .The size of the productPrices array
15.00      →   productPrices=[15.00, 300.90, 23.44]
300.90
23.44
3          →   m = 3 .The size of the productSold array
chocolate  →   productSold=[chocolate, cheese, tomato]
cheese
tomato
3          →   m = 3 .The size of the soldPrice array
15.00      →   soldPrice =[15, 300.90,10.00]
300.90
10.00
```

Sample Output 1

```
1
```

Explanation 1

```
            Price
Product     Actual   Expected  Error
chocolate    15.00    15.00
cheese      300.90   300.90
tomato       10.00    23.44    1
```

Only the *tomato* sale does not match the price list. So, the number of sale prices that were entered incorrectly is 1.

Question - 107
Two Operations Redux

You are given three integers: *x* (starting value), *y* (target value), and *z* (maximum operations). Your task is to transform *x* into *y* by performing no more than *z* operations, where each operation involves either adding 1 to *x* or subtracting 1 from *x*.

While performing these operations to reach *y*, you need to track the maximum value that *x* achieves at any point. Your goal is to maximize this value while ensuring you reach *y* within the given limit of *z* operations.

Example
x = 4
y = 4
z = 4

The maximum achievable value of x is 6. With a maximum of 4 operations available, you can add 1 twice to reach 6, then subtract 1 twice to return to 4. This approach gives you the highest possible maximum value while still reaching the target within the operations limit.

Alternative approaches would be to perform 0 operations (staying at 4) or 2 operations (add 1, then subtract 1), but these would result in lower maximum values of 4 and 5, respectively.

Function Description

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

int x: the starting value
int y: the target value
int z: the maximum number of steps

Returns

int: the maximum integer which can be made from x while converting x to y in at most z steps, or -1 if it is not possible

Constraints

- $1 \leq x, y, z \leq 10^8$

▼ Input Format For Custom Testing

The three lines contain an integer x, an integer y, and an integer z, respectively.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
8     →     x = 8
5     →     y = 5
3     →     z = 3
```

Sample Output

```
8
```

Explanation

In the first move, 8 is converted to 7, in the next move, 7 is converted to 6, and in the last move, 6 is converted to 5. Thus, the maximum value of x is 8.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4     →     x = 4
4     →     y = 4
6     →     z = 6
```

Sample Output

```
7
```

Explanation

The sequence of 6 moves is shown below:

- $4 \rightarrow 5$
- $5 \rightarrow 6$
- $6 \rightarrow 7$
- $7 \rightarrow 6$
- $6 \rightarrow 5$
- $5 \rightarrow 4$

The highest value that x can attain is 7.

Question - 108
Rolling String

You are given a string *s* consisting of lowercase English letters (a-z). You need to perform *k* operations on this string and return the resulting string.

Each operation modifies a substring of *s* by shifting each character either forward or backward in the alphabet. The operations are of two types:

1. Roll Forward (right): *i j* R
   - Every character in the substring *s[i]* to *s[j]* is replaced with the next sequential alphabetical character, for example 'a' → 'b', 'm' → 'n', 'z' → 'a'.
2. Roll Backward (left): *i j* L
   - Every character in the substring *s[i]* to *s[j]* is replaced with the preceding alphabetical character (before 'a' comes 'z'). Examples: y ← z, m ← n, a ← z

Example
s = 'abc' with the following sequential operations:

```
i       j       ch      s
                        abc
0       1       L       zac
1       2       R       zbd
0       2       R       ace
```

The variable *ch* is the direction of the roll.

Function Description
Complete the function *rollingString* in the editor with the following parameter(s):

string s: the initial string

string operations[n]: strings with three space-separated values: the integers *i* and *j*, and the character *ch*

Returns

string: the value of *s* after all operations have been performed

Constraints

- 1 ≤ length of *s* ≤ 150
- 1 ≤ *n* ≤ 100
- 0 ≤ *i* ≤ *j* < length of *s*
- *ch* is either 'L' or 'R'

▼ Input Format for Custom Testing

The first line contains a string *s*.
The next line contains an integer *n*, the size of the array *operations*.
Each of the next *n* lines contains a string *operations[i]* where 0 ≤ *i* < *n*.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function Parameters
-------     --------------------------
abc     →   s = "abc"
3       →   operations[] Size = 3
0 0 L   →   operations[] = [ "0 0 L", "2 2 L", "0 2 R"]
2 2 L
0 2 R
```

Sample Output 0

```
acc
```

Explanation 0

After performing operation "0 0 L" on "abc", *s* = "zbc"
After performing operation "2 2 L" on "zbc", *s* = "zbb"
After performing operation "0 2 R" on "zbb", *s* = "acc"

Question - 109
Creating a Binary Search Tree

Implement an algorithm that creates a Binary Search Tree from a list of unique integers. As you create the tree, count the number of times the insert function is called.

The algorithm works as follows:

- Start with an empty binary search tree.
- For each integer in the input array:
  - If the tree has no root, create a new node with the integer as the root.
  - Otherwise, insert the integer into the tree starting from the root.
  - Print the current count of insert function calls after each integer is processed.

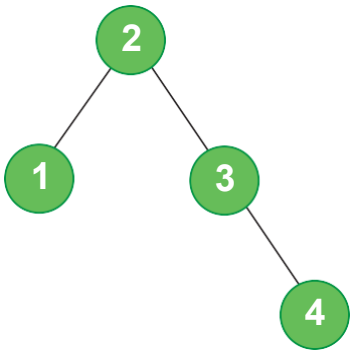
When inserting a value:

- Increment the counter tracking insert function calls.
- If the value is less than the current node:
  - If there is no left child, create a new node with the value as the left child.
  - Otherwise, recursively insert into the left subtree.
- If the value is greater than or equal to the current node:
  - If there is no right child, create a new node with the value as the right child.
  - Otherwise, recursively insert into the right subtree.

Remember that a Binary Search Tree maintains the property that all elements in a node's left subtree are less than the node's value, and all elements in the right subtree are greater than or equal to the node's value.

Example

keys = [2, 1, 3, 4]



1. First, create the root node with value 2. The insert function is not called, so *counter* = 0.
2. Insert 1 into the tree. Since 1 < 2, it becomes the left child of 2. *counter* = 1.
3. Insert 3 into the tree. Since 3 > 2, it becomes the right child of 2. *counter* = 2.
4. Insert 4 into the tree. Since 4 > 2, we go to the right child (3). Since 4 > 3, it becomes the right child of 3. This requires 2 calls to insert, so *counter* = 4.

Function Description

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

int keys[n]: the values to insert

Note: To implement your solution, you may need other variables, functions, or classes.

Returns

None. The function must print the value of *counter* after each item has been inserted into the tree, each on a separate line.

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le keys[i] \le n$
- The *keys* array contains no duplicate elements.

▼ Input Format for Custom Testing

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

▼ Sample Case 0

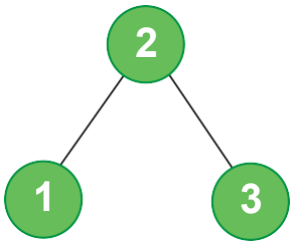
Sample Input 0

```
STDIN       Function
-----       --------
3           keys[] size n = 3
2           keys = [2, 1, 3]
1
3
```

Sample Output 0

```
0
1
2
```

Explanation 0



Perform the following sequence of insertions:

1. *key* = 2: As the tree is currently empty, we create a new node with value *2* and designate it as the root node. No calls to *insert* have occurred, so *counter* = 0.
2. *key* = 1: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's left subtree, so the value of *counter* = 1.
3. *key* = 3: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's right subtree, so the value of *counter* = 2.

▼ Sample Case 1

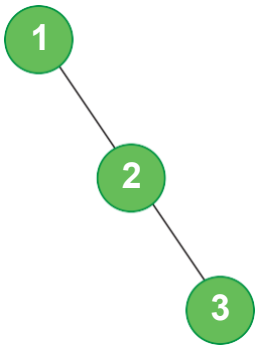
Sample Input 1

```
3
1
2
3
```

Sample Output 1

```
0
1
3
```

Explanation 1



We perform the following sequence of insertions:

1. *key* = 1: As the tree is currently empty, we create a new node with value 1 and designate it as the root node. There were no calls to *insert,* so *counter* = 0.
2. *key* = 2: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's right subtree, so the value of *counter* = 1.
3. *key* = 3: As the tree has a root node, we make a call to the *insert* function. The *insert* function runs twice before the node is inserted into the right subtree of the root's right subtree, so the value of *counter* = 3.

Question - 110
Maximum Two

Identify the subarray within the given array of integers where the bitwise AND of the first two elements is the highest after sorting the subarray in non-ascending order. Return the starting and ending indices of this subarray within the original array, with the starting index always being 0. If multiple subarrays yield the same maximal value, select the shortest one.

For instance, given the array *arr* = [5, 4, 2, 5, 7, 5], evaluate all pairs of elements to find the bitwise AND value:

5&5 = 5 4&2 = 0
5&4 = 4 4&7 = 4
5&2 = 0 2&7 = 2
5&7 = 5

The highest bitwise AND value is 5, achieved by 5 & 5 or 5 & 7. The candidate subarrays are *arr[0:3], arr[0:4]*, and *arr[0:5]*.

arr[0:3] = [5,4,2,5] = [5,5,4,2] sorted non-ascending
arr[0:4] = [5,4,2,5,7] = [7,5,5,4,2] sorted non-ascending
arr[0:5] = [5,4,2,5,7,5] = [7,5,5,5,4,2] sorted non-ascending

In each case, the bitwise AND of the first two elements equals 5. Among these, the shortest subarray is *arr[0:3]*. Hence, return [0, 3].

Function Description

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

int arr[n]: an array of integers

Returns

int[2]: the 0-based indices marking the shortest subarray

Constraints

- $2 \le n \le 5 \times 10^3$
- $1 \le arr[i] \le 10^5$, where $0 \le i < n$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN   Function
-----   -----------
2       size of arr[] n = 2
3       arr = [3, 5]
5
```

Sample Output 0

```
0
1
```

Explanation 0

Since there are only two elements, our return array must be *res* = [0,1].

▼ Sample Case 1

Sample Input 1

```
3
3
3
5
```

Sample Output 1

```
0
1
```

Explanation 1

For *arr* = *{3, 3, 5}*, *5 & 3 = 1* and *3 & 3 = 3*. We choose *arr[0:1]*, *res* = [0,1].

▼ Sample Case 2

Sample Input 2

```
3
6
3
6
```

Sample Output 2

```
0
2
```

Explanation 2

With *arr = {6, 3, 6}*, since *6 & 6 > 6 & 3*, we choose *arr[0:2]*, *res* = [0,2].

Question - 111

Distinct Digit Numbers

Given a range of integers, determine how many numbers have no repeating digits.

Example

n = 80

m = 120

The lower and upper bounds are inclusive, so there are 120 - 79 = 41 values in the range. Numbers without repeating characters are normal weight and others are bold. The two columns to the right are the valid number counts per row (normal weight) and invalid number counts (bold).

80	81	82	83	84	85	86	87	**88**	89	9	**1**
90	91	92	93	94	95	96	97	98	**99**	9	**1**
100	**101**	102	103	104	105	106	107	108	109	8	**2**
110	**111**	**112**	**113**	**114**	**115**	**116**	**117**	**118**	**119**	0	**10**
120										1	**0**

There are 27 numbers with no repeating digits, and 14 other numbers in the range. Print 27.

Function Description

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

int arr[q][2]: integer pairs representing inclusive lower *(n)* and upper *(m)* range limits

Print

For each pair *arr[i]*, print the number of integers in the inclusive range that qualify. There is no value to return from the function.

Constraints

- $1 \le q \le 10^5$
- $1 \le n \le m \le 10^6$

▼ Input Format for Custom Testing

The first line contains an integer *q*, the number of rows in the two dimensional array *arr*.

The second line contains the integer 2, the number of columns in *arr*.

Each of the next *q* lines contains two space-separated integers *n* and *m* for each *arr[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-------   -----------
```

```
2      →  arr[] size q = 2
2      →  arr[i][] size = 2 (always)
1 20   →  arr = [[1, 20], [9, 19]]
9 19
```

Sample Output 0

```
19
10
```

Explanation 0

Row 0 = [1, 20]
The set of qualifying numbers in the inclusive range between *n[0]* = 1 and *m[0]* = 20 is {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20}. This gives us *c[0]* = 19.

Row 1: [9, 19]
The set of qualifying numbers in the inclusive range between *n[1]* = 9 and *m[1]* = 19 is {9, 10, 12, 13, 14, 15, 16, 17, 18, 19}. This gives us *c[1]* = 10.

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     -----------
5      →  arr[] size q = 5
2      →  arr[i][] size = 2
7 8    →  arr = [[7, 8], [52, 80], [34, 84], [57, 64], [74, 78]]
52 80
34 84
57 64
74 78
```

Sample Output 1

```
2
26
47
8
4
```

Explanation 1

Row 0 = [7, 8]
The set of qualifying numbers in the inclusive range between *n[0]* = 7 and *m[0]* = 8 is {7, 8}. This gives us *c[0]* = 2.

Row 1 = [52, 80]
The set of qualifying numbers in the inclusive range between *n[1]* = 52 and *m[1]* = 80 is {52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 80}. This gives us *c[1]* = 26.

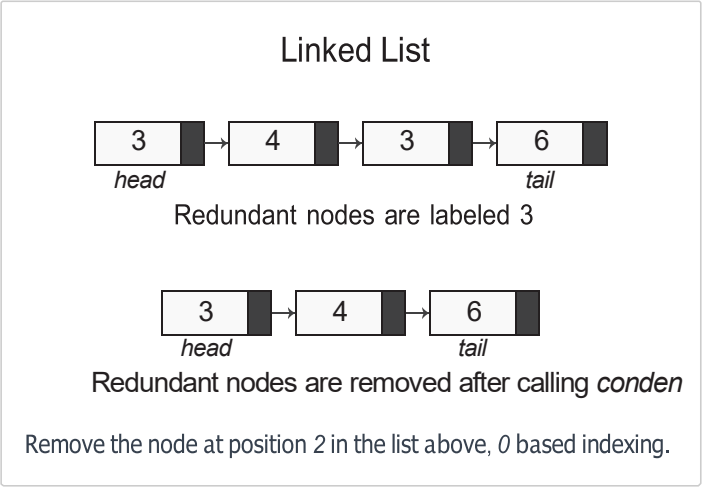
Row 2 = [34, 84]
The set of qualifying numbers in the inclusive range between *n[2]* = 34 and *m[2]* = 84 is {34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 83, 84}. This gives us *c[2]* = 47.

Row 3 = [57, 64]
The set of qualifying numbers in the inclusive range between *n[3]* = 57 and *m[3]* = 64 is {57, 58, 59, 60, 61, 62, 63, 64}. This gives us *c[3]* = 8.

Row 4 = [74, 78]
The set of qualifying numbers in the inclusive range between *n[3]* = 74 and *m[3]* = 78 is {74, 75, 76, 78}. This gives us *c[3]* = 4.

Question - 112
Condensed List

Given a list of integers, remove all nodes containing values that have appeared earlier in the list, and return a reference to the head of the modified list. For instance, in the following list, the value 3 appears as a duplicate initially:



Remove the node at position *2* in the list above, *0* based indexing.

Function Description

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

head: the head of a singly-linked list of integers, a LinkedListNode

Note:. A *LinkedListNode* has two attributes: *data*, an integer, and *next*, a reference to the next item in the list or the language equivalent of *null* at the tail.

Returns

reference to LinkedListNode: the head of the list of distinct values

Constraints

- $1 \le n \le 10^5$
- $0 \le LinkedListNode[i].val \le 1000$

▼ 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 *list*.
Each of the next *n* lines contains an integer *list[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input 0

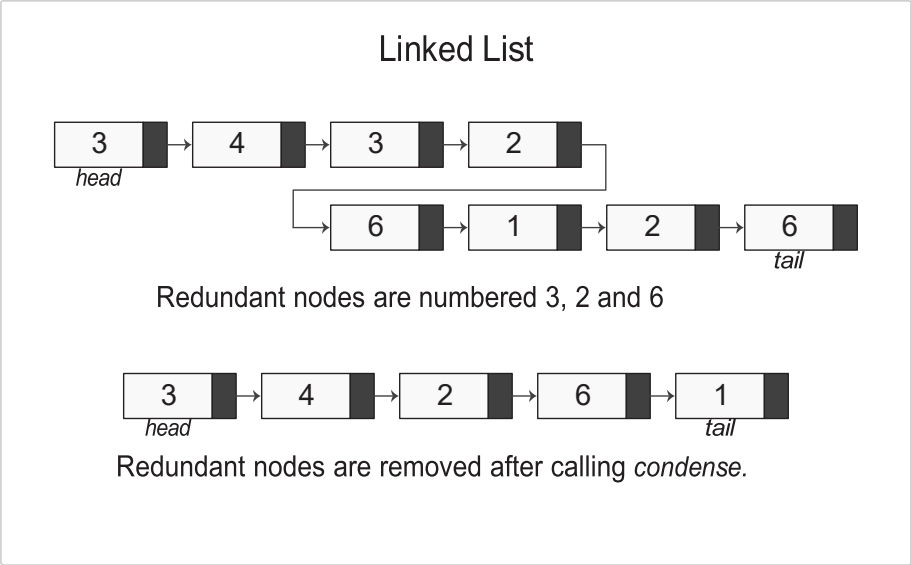
```
STDIN       Function Parameters
-------     --------------------------
8       →   list[] Size n = 8
3       →   list[] = [ 3, 4, 3, 2, 6, 1, 2, 6 ]
4
3
2
6
1
2
6
```

Sample Output 0

```
3
4
2
6
1
```

Explanation 0

The list looks like this:



From the first list in the diagram, remove:

- *list[2] = 3*
- *list[6] = 2*
- *list[7] = 6*

Question - 113

Detect the Domain Name

Find the unique domain names in HTML markup fragments. A URL is considered valid if it meets the following criteria:

- It begins with 'http://' or 'https://'.
- The protocol is followed by a host, which is a sequence of two or more period-separated labels. Each label can include lowercase letters (a-z), dashes (-), and numbers (0-9). A hostname may optionally start with the prefix 'www', 'ww2', or 'web'.
- The hostname can be followed by a path, which generally consists of segments separated by slashes.

A valid URL's *domain name* is the URL's hostname with the prefix removed. Return a list of all unique domain names found in the input.

For example, consider the following table of sample URLs and their associated domain names:

URL	Domain Name
http://hackerrank.com/	hackerrank.com
http://www.hackerrank.com/contest	hackerrank.com
http://www.xyz.com/news	xyz.com
http://web.xyz.com/about/mission	xyz.com
https://abc.xyz.com/jobs	abc.xyz.com
http://abcd.xyz.com/jobs2/	abcd.xyz.com

Note that the hostname prefixes, i.e., *www.*, *ww2.*, and *web.*, are trimmed from the domain names.

Given *n* lines of HTML markup fragments, return the unique potential domain names contained in the fragments, in alphabetical order, as a single string of semicolon-separated values. For example, using the valid URLs above, the answer is "*abc.xyz.com;abcd.xyz.com;hackerrank.com;xyz.com*".

Function Description

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

string lines[n]: an array of strings

Returns

string: all distinct valid domain names, separated by semi-colons and in alphabetical order

Constraints

- *1 ≤ n ≤ 1700*

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of lines of text.
Each of the next *n* lines contains a string that describes *lines[i]*.

▼ Sample Case 0

Sample Input 0

```
10
^ ["Train (noun)"](http://www.askoxford.com/concise_oed/train?view=uk). (definition - Compact OED). Oxford
University Press. Retrieved 2008-03-18.
^ Hello
^ World
^ C is a programming language.
^
^ Atchison, Topeka and Santa Fe Railway (1948). Rules: Operating Department. p. 7.
^ [Hydrogen trains](http://www.hydrogencarsnow.com/blog2/index.php/hydrogen-vehicles/i-hear-the-hydrogen-train-a-
comin-its-rolling-round-the-bend/)
^ [Vehicle Projects Inc. Fuel cell locomotive](http://www.bnsf.com/media/news/articles/2008/01/2008-01-09a.html)
^ Central Japan Railway (2006). Central Japan Railway Data Book 2006. p. 16.
^ ["Overview Of the existing Mumbai Suburban Railway"]
(http://web.archive.org/web/20080620033027/http://www.mrvc.indianrail.gov.in/overview.htm). _Official webpage of
Mumbai Railway Vikas Corporation_. Archived from [the original](http://www.mrvc.indianrail.gov.in/overview.htm)
on 2008-06-20. Retrieved 2008-12-11.
```

Sample Output 0

```
archive.org;askoxford.com;bnsf.com;hydrogencarsnow.com;mrvc.indianrail.gov.in
```

Explanation 0

Process the *n = 10* lines of text. Extract the unique domain names. For example, in the first line:

^ *["Train (noun)"](http://www.askoxford.com/concise_oed/train?view=uk). (definition – Compact OED). Oxford University Press. Retrieved 2008-03-18*

There is one domain name, *askoxford.com*.

Assemble the domain names into a list of distinct strings to get *{"askoxford.com", "hydrogencarsnow.com", "bnsf.com", "archive.org", "mrvc.indianrail.gov.in"}*.
Return this list as the alphabetically-ordered semicolon-delimited string.

Question - 114
Compression

Various compression methods are employed to minimize the size of messages transmitted over the internet. A specific algorithm compresses a given string by indicating the total number of consecutive occurrences of each character. For instance, consider the string "abaasass". The consecutive occurrences of each character are grouped as follows:

- *'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 occurs only once, it is added to the compressed string. If it occurs consecutively, 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*| ≤ 10^5

▼ 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"*.

Question - 115

Binary Tree Search

A binary search tree (BST) is a data structure where each node contains a value and has up to two child nodes. The left child node has a value less than its parent node, and the right child node has a value greater than or equal to its parent node. This structure allows efficient searching.

For each integer in a given list, determine if it exists in the BST. If it does, return 1; otherwise, return 0.

Function Description

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

BSTreeNode root: reference to the root node of a tree of integers
int val[q]: an array of integer items to search for

Returns

int[q]: each element *i* denotes whether *val[i]* is found in the BST

Constraints

- $1 \le n, q \le 10^5$
- $1 \le val[i] \le 5 \times 10^4$

▼ 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 number of elements in the tree.
Each of the next *n* lines contains an integer, the value of *node[i]* where $0 \le i \le n$ and *node[i]* is the root
The next line contains an integer, *q,* the number of queries
Each of the next *q* lines contains an integer to search for

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     --------
11      →   tree size n = 11
20      →   node values = [20, 10, 30, 8, 12, 25, 40, 6, 11, 13, 23]
10
30
8
12
25
40
6
11
13
23
4       →   val[] size q = 4
30      →   val = [30, 10, 12, 15]
10
12
15
```

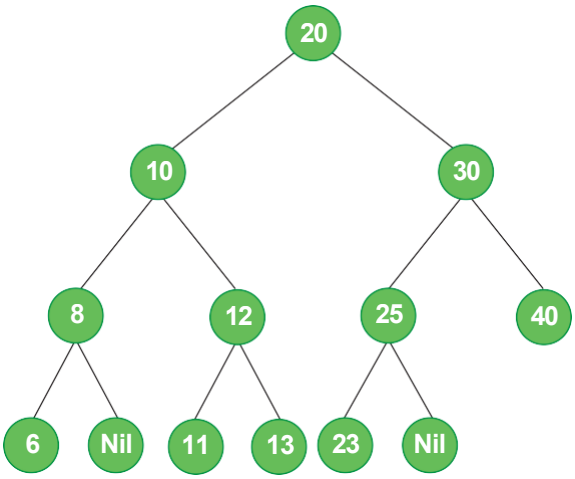
Test Values

```
30
10
12
15
```

Sample Output

```
1
1
1
0
```

Explanation



The tree is assembled as described in *Input Format for Custom Testing* by the provided code stub. Nodes marked "Nil" have no value and are placeholders to make *left* and *right* clear.

- Search for val[0] = 30. Start from the root of a tree. 30 > 20 : Search in the right subtree which has the root = 30. The item is found, return 1.
- Search for val[1] = 10. Start from the root of a tree. 10 < 20 : Search in the left subtree which has the root = 10. The item is found, return 1.
- Search for val[2] = 12. Start from the root of a tree. 12 < 20 : Search in the left subtree which has the root = 10. 12 > 10 : Search in the right subtree which has the root = 12. The item is found, return 1.
- Search for val[3] = 15. Start from the root of a tree. 15 < 20: Search in the left subtree which has the root = 10. 15 > 10 : Search in the right subtree which has the root = 12. 15 > 12 : Search in the right subtree which has the root = 13. End of the tree and the item is not found, return 0.
- The return values are [1, 1, 1, 0]

▼ Sample Case 1

Sample Input

```
STDIN       Function
-------     --------
11      →   tree size n = 11
20      →   node = [20, 10, 30, 8, 12, 25, 40, 6, 11, 13, 23]
10
30
8
12
25
40
6
11
13
23
4       →   val[] size q = 4
30      →   val = [79, 10, 20, 30, 40]
10
12
15
79
10
20
```

```
30
40
```

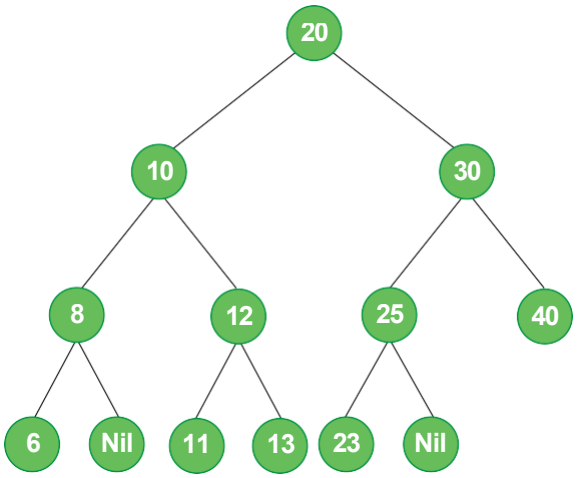
Test Values

```
79
10
20
30
40
```

Sample Output

```
0
1
1
1
1
```

Explanation



- Search for val[0] = 79. Start from the root of a tree. 79 > 20 : Search in the right subtree which has the root = 30. 79 > 30 : Search in the right subtree which has the root = 40. End of the tree and the item is not found, return 0.
- Search for val[1] = 10. Start from the root of a tree. 10 < 20 : Search in the left subtree which has the root = 10. The item is found, return 1.
- Search for val[2] = 20. Start from the root of a tree. 20 = 20 : The item is found, return 1.
- Search for val[3] = 30. Start from the root of a tree. 30 > 20 : Search in the right subtree which has the root = 30. The item is found, return 1.
- Search for val[4] = 40. Start from the root of a tree. 40 > 20 : Search in the right subtree which has the root = 30. 40 > 30: Search in the right subtree which has the root = 40. The item is found, return 1.
- The return values are [0, 1, 1, 1, 1]

Question - 116
Ancestral Names

Given a list of strings containing a name and a Roman numeral, sort the list:

- First by name
- Then by the decimal value of the Roman numeral

Roman numerals follow these rules:

- A value is not repeated more than three times
- When a smaller value precedes a larger value, it indicates subtraction
- I, II, III, IV, V, VI, VII, VIII, IX, and X represent 1 through 10

- XX, XXX, XL, and L represent 20, 30, 40, and 50
- For any other two-digit number < 50, concatenate the Roman numeral(s) that represent its multiples of ten with the Roman numeral(s) for its values < 10. For example, 43 is 40 + 3 = 'XL' + 'III' = 'XLIII'.

Example

Input: *names* = ['Steven XL', 'Steven XVI', 'David IX', 'Mary XV', 'Mary XIII', 'Mary XX']

Output: ['David IX', 'Mary XIII', 'Mary XV', 'Mary XX', 'Steven XVI', 'Steven XL']

With decimal values, the sorted list is ['David 9', 'Mary 13', 'Mary 15', 'Mary 20', 'Steven 16', 'Steven 40'].

Function Description

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

string names[n]: names with roman numerals

Returns

string[n]: an array of strings sorted first by given name, then by ordinal

Constraints

- $1 \le n \le 50$
- Each *names[i]* is a single string composed of *2* space-separated values: *givenName* and *romanNumeral*.
- *romanNumeral* represents a number between *1* and *50*, inclusive.
- $1 \le |givenName| \le 20$
- Each *givenName* starts with an uppercase letter *ascii[A-Z]* which is followed by lowercase letters *ascii[a-z]*.
- Each *names[i]* is distinct.

▼ Input Format for Custom Testing

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

Each of the next *n* lines contains an element *names[i].*

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
2   →  names[] size n = 2
Louis IX   →  names = ['Louis IX', 'Louis VIII']
Louis VIII
```

Sample Output

```
Louis VIII
Louis IX
```

Explanation

Sort first by *givenName* and then, if *givenName* is not unique, by the value of the Roman numeral. In decimal, the list is sorted ['Louis 8', 'Louis 9'].

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
2   →  names[] size n = 2
Philippe I   →  names = ['Philippe I', 'Philip II']
Philip II
```

Sample Output

```
Philip II
```

```
Philippe I
```

'Philippe' is alphabetically greater than 'Philip'.

Question - 117
Are They Pangrams

Determine which strings in a given list are pangrams. A pangram contains every letter of the English alphabet from 'a' to 'z'.

In the return array, for each string in the input list, store "1" if it is a pangram and "0" if it is not.

Example
pangram = ['pack my box with five dozen liquor jugs', 'this is not a pangram']

- The string *'pack my box with five dozen liquor jugs'* is a pangram because it contains all the letters 'a' through 'z'.
- The string *'this is not a pangram'* is not a pangram.
- Assemble a string of the two results, in order. The result is *'10'*.

Function Description
Complete the function *isPangram* in the editor with the following parameter(s):
string pangram[n]: the sentences to check

Returns
string: a string where each position represents the results of a test. Use '*1*' for true and '*0*' for false.

Constraints
- $1 \le n \le 100$
- Each string *pangram[i]* (where $0 \le i < n$) is composed of lowercase letters and spaces.
- $1 \le$ length of *pangram[i]* $\le 10^5$

▼ Input Format for Custom Testing
The first line contains an integer *n*, the size of the array *pangram*.
The next *n* lines each contain an element, *pangram[i]*, where $0 \le i < n$.

▼ Sample Case 0
Sample Input 0

```
STDIN                                                    Function Parameters
-----                                                    -------------------
4                                                   →    pangram[] size n = 4
we promptly judged antique ivory buckles for the next prize   →    pangram[] = ["we promptly judged antique ivory
buckles for the next prize",
we promptly judged antique ivory buckles for the prizes                    "we promptly judged antique ivory
buckles for the prizes",
the quick brown fox jumps over the lazy dog                                "the quick brown fox jumps over the
lazy dog",
the quick brown fox jump over the lazy dog                                 "the quick brown fox jump over the
lazy dog" ]
```

Sample Output 0

```
1010
```

Explanation 0

```
pangram[0] = True
pangram[1] = False
pangram[2] = True
pangram[3] = False
```

- The strings *pangram[0]* and *pangram[2]* are pangrams, and the others are not.
- The result is *'1010'*

▼ Sample Case 1

```
STDIN                                                                  Function Parameters
-----                                                                  -------------------
4                                                                    → pangram[] Size n = 4
cfchcfcvpalpqxenhbytcwazpxtthjumliiobcznbefnofyjfsrwfecxcbmoafes tnulqkvx
oxhctvhybtikkgeptqulzukfmmavacshugpouxoliggcomykdnfayayqutgwivwldrkp
gpecfrak zzaxrigltstcrdyhelhz rasrzibduaq cnpuommogatqem
hbybsegucruhxkebrvmrmwhweirx mbkluwhfapjtga liiylfphmzkq
```

Sample Output 1

```
0000
```

Explanation 1

```
pangram[0] = False
pangram[1] = False
pangram[2] = False
pangram[3] = False
```

- No string is a pangram.
- The result is *'0000'*

Question - 118
Detecting Valid Latitude and Longitude Pairs

Create a function that checks if strings in a given list represent valid latitude and longitude pairs.
For each string in the list, the function should determine if it is a valid coordinate pair and print "Valid" or "Invalid" based on the following criteria:
A string *(X, Y)* is considered valid if:

- It starts with '(', contains a comma after *X*, and ends with ')'
- There is no space between the opening parenthesis and the first character of *X*
- There is no space between the comma and the last character of *X*
- There is exactly one space between the comma and the first character of *Y*
- There is no space between *Y* and the closing parenthesis
- *X* and *Y* are decimal numbers and may have a sign (+ or -)
- There are no leading zeros in *X* or *Y*
- No other characters are allowed in *X* or *Y*
- -90 ≤ *X* ≤ 90 and -180 ≤ *Y* ≤ 180

Example

coordinates = ['(90, 180)', '(+90, -180)', ' (90, 180)', '(90.0, 180.1)', '(85S, 95W)']

```
coordinates      results  Reasoning
(90, 180)        Valid
(+90, -180)      Valid
 (90, 180)       Invalid  Preceding space
```

```
(90.0, 180.1)   Invalid  Y is out of bounds
(85S, 95W)      Invalid  Extraneous characters
```

Function Description

Complete the function *isValid* in the editor below. The function must print the results of each test on a new line.

isValid has the following parameter(s):

string coordinates[n]: the strings to test

Constraints

1 ≤ n ≤ 100

▼ 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 *coordinates*.

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

▼ Sample Case 0

Sample Input 0

```
12
(75, 180)
(+90.0, -147.45)
(77.11112223331, 149.99999999)
(+90, +180)
(90, 180)
(-90.00000, -180.0000)
(75, 280)
(+190.0, -147.45)
(77.11112223331, 249.99999999)
(+90, +180.2)
(90., 180.)
(-090.00000, -180.0000)
```

Sample Output 0

```
Valid
Valid
Valid
Valid
Valid
Valid
Invalid
Invalid
Invalid
Invalid
Invalid
Invalid
```

Explanation 0

The first six pairs are valid because *X* and *Y* satisfy the criteria for numerical values and formats.

The next four have numbers outside the numerical ranges for *X* or *Y*.

(90., 180.) is invalid because of an extra decimal point (.) after *90* and *180*. *(-090.0000, -180.0000)* is invalid because of the leading zero before *90*.

Question - 119

Two Circles

Determine the relationship between two circles on a Cartesian plane.

Each circle is defined by three parameters:

- *X*: the x-coordinate of the center
- *Y*: the y-coordinate of the center
- *R*: the radius

Two circles *A* and *B* will be centered either on the X-axis ($Y_A = Y_B = 0$) or on the Y-axis ($X_A = X_B = 0$), but not both.

Two circles can have one of these relationships:

- "Touching": they touch at a single point
- "Concentric": they have the same center point
- "Intersecting": they touch at two points
- "Disjoint-Outside": one exists outside the other without touching
- "Disjoint-Inside": one is contained inside the other (not concentric)

Example

circlePairs = ['3 0 10 5 0 3', '0 1 4 0 1 5']

First pair: '3 0 10 5 0 3' represents Circle *A*(3,0,10) and Circle *B*(5,0,3)

- Circle *A* is centered at (3,0) with radius 10, extending from -7 to 13 on x-axis
- Circle *B* is centered at (5,0) with radius 3, extending from 2 to 8 on x-axis
- The relationship is "Disjoint-Inside"

Second pair: '0 1 4 0 1 5' represents Circle *A*(0,1,4) and Circle *B*(0,1,5)

- Circle *A* is centered at (0,1) with radius 4, extending from -3 to 5 on y-axis
- Circle *B* is centered at (0,1) with radius 5, extending from -4 to 6 on y-axis
- The relationship is "Concentric"

Function Description

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

string circlePairs[n]: each string contains six space-separated integers. The first three integers are *X, Y,* and *R* for circle *A*, and the last three are *X, Y,* and *R* for circle *B.*

Returns

string [n]: each string *i* is the relation between the circles described in *circlePairs[i]*

Constraints

- $1 \le n \le 5000$
- $0 \le X, Y, R \le 5000$

▼ Input Format for Custom Testing

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

Each of the next *n* lines contains a string of six space-separated integers, *circlePairs[i]*. The first three integers are *X, Y,* and *R* for circle *A* and the last three are *X, Y,* and *R* for circle *B*.

▼ Sample Case 0

Sample Input

```
STDIN                Function
-------              --------
4                  → circlePairs[] size n = 4
12 0 21 14 0 23    → circlePairs = ['12 0 21 14 0 23', '0 45 8 0 94 9', '35 0 13 10 0 38', '0 26 8 0 9 25']
0 45 8 0 94 9
35 0 13 10 0 38
0 26 8 0 9 25
```

Sample Output

```
Touching
Disjoint-Outside
Touching
Touching
```

Explanation

1. The circles touch at (-9, 0).
   - Circle *A* is centered at (12, 0) and extends along the x-axis from -9 to 33.
   - Circle *B* is centered at (14, 0) and extends along the x-axis from -9 to 37.
2. The circles share no points in common.
   - Circle *A* is centered at (0, 45) and extends along the y-axis from 37 to 53.
   - Circle *B* is centered at (0, 94) and extends along the y-axis from 85 to 103.
3. The circles touch at (48, 0).
   - Circle *A* is centered at (35, 0) and extends along the x-axis from 22 to 48.
   - Circle *B* is centered at (10, 0) and extends along the x-axis from -28 to 48.
4. The circles touch at (0, 34).
   - Circle *A* is centered at (0, 26) and extends along the y-axis from 18 to 34.
   - Circle *B* is centered at (0, 9) and extends along the y-axis from -15 to 34.

The array *result* = ["Touching", "Disjoint-Outside", "Touching", "Touching"] is returned.

▼ Sample Case 1

Sample Input

```
STDIN                Function
-------              -------
5                  → circlePairs[] size n = 5
0 5 9 0 9 7        → circlePairs = ['0 5 9 0 9 7', '0 15 11 0 20 16', '26 0 10 39 0 23', '37 0 5 30 0 11', '41 0 0
28 0 13']
0 15 11 0 20 16
26 0 10 39 0 23
37 0 5 30 0 11
41 0 0 28 0 13
```

Sample Output

```
Intersecting
Touching
Touching
Intersecting
Touching
```

Explanation

1. The circles intersect.
   - Circle *A* is centered at (0, 5) and extends along the y-axis from -4 to 14.
   - Circle *B* is centered at (0, 9) and extends along the y-axis from 2 to 16.
2. The circles touch at (41, 0).
   - Circle *A* is centered at *(41, 0) and extends along the x-axis from 41 to 41.*
   - Circle *B* is centered at *(28, 0) and extends along the x-axis from 15 to 41.*
3. The circles touch at (0, 4).
   - Circle *A* is centered at (0, 15) and extends along the y-axis from 4 to 26.
   - Circle *B* is centered at *(0, 20) and extends along the y-axis from 4 to 36.*
4. The circles touch at (16,0).
   - Circle *A* is centered at (26, 0) and extends along the x-axis from 16 to 36.
   - Circle *B* is centered at *(39, 0) and extends along the x-axis from 16 to 62.*
5. The circles intersect.
   - Circle *A* is centered at *(37, 0) and extends along the x-axis from 32 to 42.*

- Circle *B* is centered at *(30, 0) and extends along the x-axis from 19 to 41.*

6. The circles touch at (41,0).
   - Circle *A* is centered at *(41, 0) and extends along the x-axis from 41 to 41.*
   - Circle *B* is centered at *(28, 0) and extends along the x-axis from 15 to 41.*

Question - 120
Two Strings

Given two arrays of strings, check if corresponding elements (at the same index) in each array contain at least one common substring. Return an array where for each pair, if there is a common substring, print YES, or NO if there is not.

Example

a = ["ab", "cd", "ef"]
b = ["af", "ee", "ef"]
Analysis:

- *i* = 0: *a[0]* = "ab", *b[0]* = "af" → They share substring "a" → print YES
- *i* = 1: *a[1]* = "cd", *b[1]* = "ee" → No common substring → print NO
- *i* = 2: *a[2]* = "ef", *b[2]* = "ef" → They share substring "ef" → print YES

Function Descriptio

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

string a[n]: an array of strings

string b[n]: an array of strings

Returns

void: YES or NO 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].
- length of *a* = length of b
- 1 ≤ length of *a,* length of b ≤ 10^3
- 1 ≤ length of *a[i],* length of b[i] ≤ 10^4

▼ Input Format for Custom Testing

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'.

Question - 121
Is Possible

Given a pair of integers (a, b), you can perform the following operations any number of times and in any order:

- *(a, b) → (a + b, b)*
- *(a, b) → (a, a + b)*

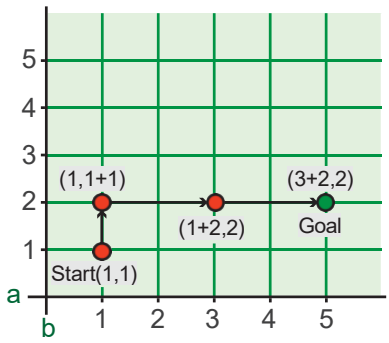
Return a string, 'Yes' or 'No', indicating whether (a, b) can be converted to (c, d) using the specified operations.

Example

(a, b) = (1, 1)

(c, d) = (5, 2)

Perform the operations *(1, 1 + 1)* to get *(1, 2)*, *(1 + 2, 2)* to get *(3, 2)*, and *(3+2, 2)* to get *(5, 2)*. Alternatively, the first operation could be *(1+1, 1)* to get *(2, 1)* and so on. The diagram below demonstrates the example that represents the pairs as Cartesian coordinates:



Function Description

Complete the function *isPossible* with the following parameter(s):

int a: first value in *(a, b)*

int b: second value in *(a, b)*

int c: first value in *(c, d)*

int d: second value in *(c, d)*

Returns:

str: Return *'Yes'* if *(a, b)* can be converted to *(c, d)* by performing zero or more of the operations specified above, or *'No'* if not.

Constraints

- *1 ≤ a, b, c, d ≤ 1000*

▼ Input Format for Custom Testing

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

The first line contains an integer, *a*.
The next line contains an integer, *b*.
The next line contains an integer, *c*.
The next line contains an integer, *d*.

▼ Sample Case 0

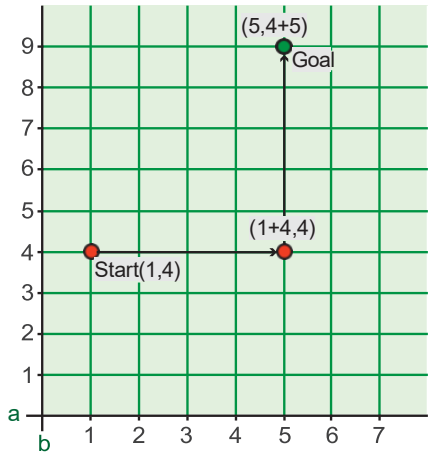
Sample Input 0

```
STDIN       Function
-----       --------
1       →   a = 1
4       →   b = 4
5       →   c = 5
9       →   d = 9
```

Sample Output 0

```
Yes
```

Explanation 0



Convert *(1, 4)* to *(5, 9)* by performing the following sequence of operations: *(1, 4)* → *(5, 4)* → *(5, 9)*.

▼ Sample Case 1

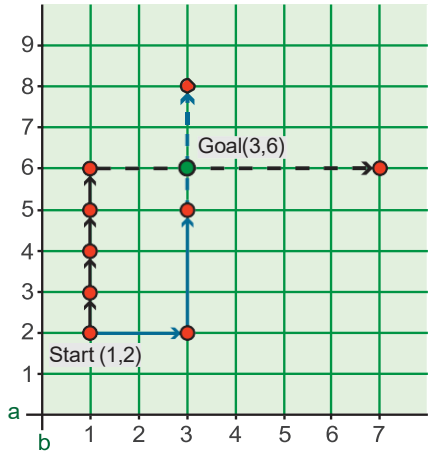
Sample Input 1

```
STDIN       Function
-----       --------
1       →   a = 1
2       →   b = 2
3       →   c = 3
6       →   d = 6
```

Sample Output 1

```
No
```

Explanation 1



Attempt to convert *(1, 2) → (3, 6)*. The graph shows some possible paths toward the goal. The dashed lines indicate next moves, which miss the destination. Return *No* as the answer.

Question - 122
Closest Color

A pixel color is defined as a 24-bit integer, with 8 bits each for red, green, and blue components (RGB). Each component has an intensity value between 0 (low) and 255 (high).

The distance between two pixels with RGB values *(r1, g1, b1)* and *(r2, g2, b2)* is calculated as:

$$d = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2}$$

For reference, the RGB values are defined as follows:

Pure Color	R	G	B
Black	0	0	0
White	255	255	255
Red	255	0	0
Green	0	255	0
Blue	0	0	255

Given a 24-bit binary string representing a pixel, identify which of these five pure colors it is closest to. If there is more than one closest color, return "Ambiguous".

Example

n = 1

pixels = ["000000001111111100000110"]

The pixel's RGB value is (0, 255, 6) :

1. red = $(00000000)_2 = (0)_{10}$
2. green = $(11111111)_2 = (255)_{10}$
3. blue = $(00000110)_2 = (6)_{10}$

Calculate its Euclidean distance to each color:

```
Pure Black: d = ((0 -   0)^2 + (255 -   0)^2 + (6 -   0)^2)^1/2 =  65061^1/2 = 255.0705785
Pure White: d = ((0 - 255)^2 + (255 - 255)^2 + (6 - 255)^2)^1/2 = 127026^1/2 = 356.4070706
Pure Red:   d = ((0 - 255)^2 + (255 -   0)^2 + (6 -   0)^2)^1/2 = 130086^1/2 = 360.6743684
Pure Green: d = ((0 -   0)^2 + (255 - 255)^2 + (6 -   0)^2)^1/2 =     36^1/2 = 6
Pure Blue:  d = ((0 -   0)^2 + (255 -   0)^2 + (6 - 255)^2)^1/2 = 127026^1/2 = 356.4070706
```

The color with the smallest distance to the pixel is Pure Green, so the answer is "Green".

Function Description

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

string pixels[n]: an array of 24-bit binary strings representing pixels as described

Returns

string[n]: each element *i* represents the closest color for its associated *pixels[i]*

Constraints

- $1 \le n \le 100$

▼ Input Format for Custom Testing

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

Each of the next *n* lines contains a 24 character bit-string *pixels[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input 0

```
STDIN                       Function
-----                       --------
5                       →   pixels[] size n = 5
101111010110011011100100 →  pixels= ["101111010110011011100100", "110000010101011111101111",
"100110101100111111101101",  "010111011010010110000011",
110000010101011111101111              "000000001111111111111111"]
100110101100111111101101
010111011010010110000011
000000001111111111111111
```

Sample Output 0

```
White
White
White
Green
Ambiguous
```

Explanation 0

Process the following *n* = 5 binary strings:

0. 101111010110011011100100 → (189, 102, 228) is closest to White:
   - The distance to pure White is 168.80165875962237.
   - The distance to pure Blue is 216.45784809056934.
   - The distance to pure Red is 258.3486016993318.
   - The distance to pure Black is 313.22356233208257.
   - The distance to pure Green is 333.33766663850037.
1. 110000010101011111101111 → (193, 87, 239) is closest to White:
   - The distance to pure White is 179.78876494375282.
   - The distance to pure Blue is 212.30638238168913.
   - The distance to pure Red is 261.7899921692959.
   - The distance to pure Black is 319.2788749666974.
   - The distance to pure Green is 350.13425996323184.
2. 100110101100111111101101 → (154, 207, 237) is closest to White:
   - The distance to pure White is 113.26517558367179.

- The distance to pure Blue is 258.62907802488104.
- The distance to pure Green is 286.6862396418775.
- The distance to pure Red is 330.4829798945779.
- The distance to pure Black is 350.334126228091.

3. 010111011010010110000011 → (93, 165, 131) is closest to Green:
   - The distance to pure Green is 184.14668066516975.
   - The distance to pure White is 222.97981971469974.
   - The distance to pure Blue is 226.38462845343543.
   - The distance to pure Black is 230.2932912613826.
   - The distance to pure Red is 265.7630523605567.
4. 000000001111111111111111 → (0, 255, 255) is equidistant from White, Green, and Blue, so it is Ambiguous:
   - The distance to pure White is 255.0.
   - The distance to pure Green is 255.0.
   - The distance to pure Blue is 255.0.
   - The distance to pure Black is 360.62445840513925.
   - The distance to pure Red is 441.6729559300637.

Return the array ["White", "White", "White", "Green", "Ambiguous"] as the answer.

Question - 123
Preprocess Dates

Users enter dates as strings in a web form. These dates need to be converted to a standard format before being stored in the database. Write a function to perform this conversion.

Given a date string in the format "Day Month Year", where:

- Day is a string like "1st", "2nd", "3rd", "21st", "22nd", "23rd", "31st", or a number followed by "th" (such as "4th" or "12th")
- Month is the first three letters of the month name in English, from "Jan" for January to "Dec" for December
- Year is a 4-digit number ranging from 1900 to 2100

Convert the date string "Day Month Year" to the format "YYYY-MM-DD" (4-digit year, 2-digit month, 2-digit day).

Example

- *1st Mar 1974 → 1974-03-01*
- *22nd Jan 2013 → 2013-01-22*
- *7th Apr 1904 → 1904-04-07*

Function Description

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

string dates[n]: date strings in the format *Day Month Year*

Returns:

string[n]: array of converted date strings

Constraints

- The values of *Day*, *Month*, and *Year* are restricted to the value ranges specified above.
- The given dates are guaranteed to be valid, so no error handling is necessary.
- $1 \le n \le 10^4$

▼ 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 *dates*.

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

▼ Sample Case 0

Sample Input 0

```
STDIN               Function
-----               --------
10              →   dates[] size n = 10
20th Oct 2052   →   dates = ["20th Oct 2052", "6th Jun 1933", "26th May 1960", "20th Sep 1958", "16th Mar 2068",
"25th May 1912", "16th Dec 2018",
6th Jun 1933                "26th Dec 2061", "4th Nov 2030", "28th Jul 1963"]
26th May 1960
20th Sep 1958
16th Mar 2068
25th May 1912
16th Dec 2018
26th Dec 2061
4th Nov 2030
28th Jul 1963
```

Sample Output 0

```
2052-10-20
1933-06-06
1960-05-26
1958-09-20
2068-03-16
1912-05-25
2018-12-16
2061-12-26
2030-11-04
1963-07-28
```

Explanation

The conversions are:

```
20th Oct 2052 → 2052-10-20
 6th Jun 1933 → 1933-06-06
26th May 1960 → 1960-05-26
20th Sep 1958 → 1958-09-20
16th Mar 2068 → 2068-03-16
25th May 1912 → 1912-05-25
16th Dec 2018 → 2018-12-16
26th Dec 2061 → 2061-12-26
 4th Nov 2030 → 2030-11-04
28th Jul 1963 → 1963-07-28
```

Question - 124

Simple Customer Support Ticketing

A support system uses braces to represent ticket statuses. Open braces ('(', '[', '{') represent open tickets, and closed braces (')', ']', '}') represent closed tickets. Braces in a string are considered balanced if they meet these conditions:

- All open braces must be closed.
- Each closed brace must have a matching open brace.
- Any set of nested braces must be closed before its surrounding braces.

Given an array of strings containing braces, verify if the braces in each string are balanced. Return "YES" if balanced and "NO" otherwise.

Example

braces = ["[{}]", "{[}]"]

- The first string "[{}]" is balanced because all braces are properly closed and nested.

- The second string "[{}" is not balanced because the inner brace '{' was not closed before its surrounding '['.

The result is ["YES", "NO"].

Function Description

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

string braces[n]: the strings to analyze

Returns

string[n]: either "YES" or "NO", where the string at each index *i* denotes whether the braces are balanced in *braces[i].*

Constraints

- $1 \le n \le 15$
- 1 ≤ length of each *braces[i]* ≤ 100
- Each *braces[i]* consists of (,), {, }, [, and] only.

▼ Input Format For Custom Testing

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

Each of the next *n* lines contains a string that describes *braces[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-------     --------
2       →   braces[] size n = 2
{}[]() →    braces = ['{}[]()', '{[]}']
{[]}
```

Sample Output

```
YES
NO
```

Explanation

- The braces in the first string "{}[]()" are balanced, because all braces are closed
- The braces in the second string "{[}]" are not balanced, because the nested braces "{[" were not closed in order "}]" and not all open and closed braces match.

Question - 125

Co-prime Count

Two integers are considered coprimes if their greatest common divisor (GCD) is 1.

Given an array of positive integers *A*, return an array *B* where *B[i]* represents the count of integers *j* such that:

- $1 \le j \le A[i]$
- *j* is coprime with *A[i]*

Example

A = [5, 8, 14]

The number *A[0] = 5* is prime. All numbers greater than zero and less than *5*, i.e., 1-4, are co-prime to *5*, so *B[0] = 4*.

For *A[1] = 8*, the integers *[2, 4, 6]* share at least the common divisor of *2* with *8*. The 4 values, *[1, 3, 5, 7]*, are co-primes, so *B[1] = 4*.

For *A[2] = 14*, the integers *[2, 4, 6, 7, 8, 10, 12]* share a common divisor > 1 with 14. The 6 co-primes are *[1, 3, 5, 9, 11, 13]*, so *B[2] = 6*.

The return array is *B = [4, 4, 6]*.

Function Description

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

int A[n]: an array of integers

Return

int[n]: the number of co-primes for each test case

Constraints

$1 \le A[i] \le 10^5$

$1 \le size\ of\ A \le 10^5$

▼ 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 an integer *A[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-----     -----------
1     →   A[] size n = 1
1     →   A = [1]
```

Sample Output 0

```
1
```

Explanation 0

1 is the only number that satisfies the given property, since the greatest common divisor of (1,1) = 1

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     -----------
2     →   A[] size n = 2
1     →   A = [1, 3]
3
```

Sample Output 1

```
1
2
```

Explanation

B[0] = count of co-primes of A[0] = 1 in [1] = 1 *B[1]* = count of co-primes of *A[1]* = *3* in *[1,2,3]* = *2* because *gcd(1,3)* = *1, gcd(2,3)* = *1* and *gcd(3,3)* = *3*.

Question - 126

Is the Number Present?

You are given an array of integers. For each element, determine if that value appears earlier in the array and if it appears later in the array. Create two binary strings to represent these occurrences:

- First string: For each position, write '1' if the value at that position also appears earlier in the array, or '0' if not

- Second string: For each position, write '1' if the value at that position appears later in the array, or '0' if not

Return an array containing these two strings.

Example

num = [1, 2, 3, 2, 1]

```
                                          Bit Value
i  num[i]  num[0:i-1]      num[i+1:n-1] string 1 string 2
-  ------  ------------    ------------ -------- --------
0  1       NULL            [2,3,2,1]       0        1
1  2       [1]             [3,2,1]         0        1
2  3       [1,2]           [2,1]           0        0
3  2       [1,2,3]         [1]             1        0
4  1       [1,2,3,2]       NULL            1        0
```

- For *i* = 0, the value 1 does not occur earlier in the array, but it does occur later. The two strings are now ["0", "1"].
- For *i* = 1, the value 2 does not occur earlier in the array, but it does occur later. The two strings are now ["00", "11"].
- For *i* = 2, the value 3 does not occur earlier or later in the array. The two strings are now ["000", "110"].

After similar analyses on the remaining values, the return array is ["00011", "11000"].

Function Description

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

int num[n]: an array of integers

Returns

string[2]: an array of 2 strings as described

Constraints

$1 \le n \le 10^4$

$0 \le num[i] \le 10^4$

▼ Input Format for Custom Testing

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

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

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-----     -----------
6     →   num[] length n = 6
1     →   num = [1, 3, 2, 3, 4, 1]
3
2
3
4
1
```

Sample Output 0

```
000101
110000
```

Explanation 0

```
                                          Bit Value
i  num[i]  num[0:i-1]      num[i+1:n-1]  string 1    string 2
-  ------  ------------    ------------  ----------  ----------
0  1       NULL            [3,2,3,4,1]       0           1
```

```
1  3        [1]              [2,3,4,1]        0          1
2  2        [1,3]            [3,4,1]          0          0
3  3        [1,3,2]          [4,1]            1          0
4  4        [1,3,2,3]        [1]              0          0
5  1        [1,3,2,3,4]      NULL             1          0
```

Question - 127

Factorial Remainder

Given an integer *n*, find out how many positive integers less than or equal to *n* satisfy the condition (*x*-1)! % *x* = *x*-1 where % is the modulo operator. The values tested are $1 \le x \le n$.

Note:

```
x!  = x × x-1 × x-2 × ..... × 1
0! = 1
```

For example, the following shows the calculations for *x* values up to 15, cumulating *count*:

```
x: 1,  (x-1)! = 0! = 1,  1%1 = 0
count = 1
x: 2,  (x-1)! = 1! = 1,  1%2 = 1
count = 2
x: 3,  (x-1)! = 2! = 2,  2%3 = 2
count = 3
x: 4,  (x-1)! = 3! = 6,  6%4 = 2
x: 5,  (x-1)! = 4! = 24,  24%5 = 4
count = 4
x: 6,  (x-1)! = 5! = 120,  120%6 =
0
x: 7,  (x-1)! = 6! = 720,  720%7 =
6
count = 5
x: 8,  (x-1)! = 7! = 5040,  5040%8 = 0
x: 9,  (x-1)! = 8! = 40320,  40320%9 = 0
x: 10,  (x-1)! = 9! = 362880,  362880%10 = 0
x: 11,  (x-1)! = 10! = 3628800,  3628800%11 = 10
count = 6
x: 12,  (x-1)! = 11! = 39916800,  39916800%12 = 0
x: 13,  (x-1)! = 12! = 479001600,  479001600%13 = 12
count = 7
x: 14,  (x-1)! = 13! = 6227020800,  6227020800%14 = 0
x: 15,  (x-1)! = 14! = 87178291200,  87178291200%15 = 0
```

Function Description

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

n: an integer

Returns

int: the number of values *x* such that $1 \le x \le n$ for which the condition is true

Constraints

$1 \le n \le 10^5$

▼ Input Format for Custom Testing

The only line contains an integer *n*.

▼ Sample Case 0

Sample Input 0

```
2
```

Sample Output 0

```
2
```

Explanation 0

x = 1, (1-1)! % 1 = 1 % 1 = 0 = (1-1)

x = 2, (2-1)! % 2 = 1 % 2 = 1 = (2-1), hence 2.

▼ Sample Case 1

Sample Input 1

```
4
```

Sample Output #2:

```
3
```

Explanation #2:

For x = 1 and x = 2, see above. Cumulative count is 2 so far.

x = 3, (3-1)! % 3 = 2 % 3 = 2 = (3-1). Cumulative count is 3.

x = 4, (4-1)! % 4 = 6 % 4 = 2 ≠ (4-1). Cumulative count remains 3.

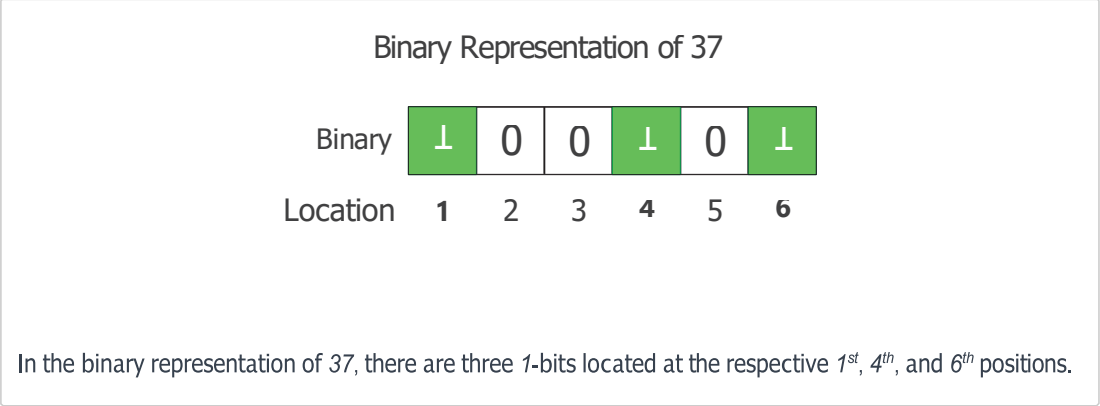
Question - 128

Counting Bits

Given an integer, *n*, determine the following:

1. How many 1-bits are present in its binary representation?
2. What are the positions of each 1-bit, listed in ascending order?

For example, this diagram represents *n = 37*:



In the binary representation of *37*, there are three *1*-bits located at the respective 1^{st}, 4^{th}, and 6^{th} positions.

Note: The leftmost 1 bit is always considered position 1, and preceding zeros are ignored when determining positions.

Function Description

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

n: an integer

Returns

int *results[]:* the number of *1*'s stored at *results[0],* followed by the positions of all *1*'s in its binary representation in ascending order

Constraints

- $1 < n < 10^9$

▼ Input Format for Custom Testing

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

The single input is an integer, *n*.

▼ Sample Case 0

Sample Input

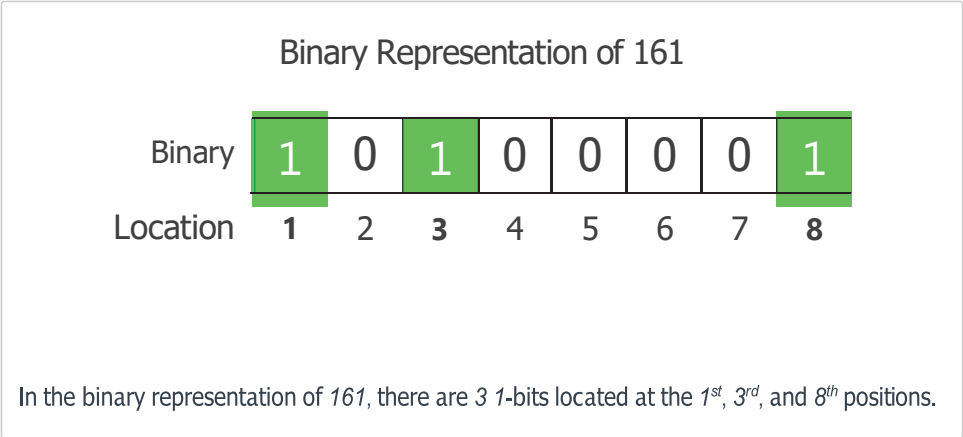
```
161
```

Sample Output

```
3
1
3
8
```

Explanation

The integer $n = (161)_{10}$ converts to $(10100001)_2$:



In the binary representation of *161*, there are *3 1*-bits located at the 1^{st}, 3^{rd}, and 8^{th} positions.

Because there are three *1*-bits, the return array is *3 + 1 = 4* units in length. Store the *1*'s count, *3*, at index *0*. Then store the locations of the *1*-bits in order, low to high. Return the array *[3, 1, 3, 8]* as the answer.

Question - 129

Romanizer

The following table contains some reference values for converting between Arabic numerals and Roman numerals:

Arabic	Roman	Arabic	Roman
1	I	40	XL
2	II	50	L
3	III	90	XC
4	IV	100	C
5	V	400	CD
6	VI	500	D
7	VII	900	CM
8	VIII	1000	M
9	IX		
10	X		

Given an integer, convert it to its Roman numeral equivalent.

Example

numbers = [1, 49, 23]

- 1 is represented as 'I'

- 49 is 40 + 9, so 'XLIX'
- 23 is 'XXIII'

The return array is ["I", "XLIX", "XXIII"].

Function Description

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

int numbers[n]: an array of integers

Returns

string[n]: the integers as their Roman numeral equivalents

Constraints

- $1 \le n \le 1000$
- $1 \le numbers[i] \le 1000$

▼ Input Format for Custom Testing

The first line contains an integer *n*, the size of the array *numbers*.
Each of the next *n* lines contains an integer *numbers[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN     Function
-------   -----------
5     →   numbers[] size n = 5
1     →   numbers = [1, 2, 3, 4, 5]
2
3
4
5
```

Sample Output 0

```
I
II
III
IV
V
```

Explanation 0

We perform the following conversions on the array *[1, 2, 3, 4, 5]*:

0. *numbers[0]* = 1 corresponds to Roman numeral 'I'.
1. *numbers[1]* = 2 corresponds to Roman numeral 'II'.
2. *numbers[2]* = 3 corresponds to Roman numeral 'III'.
3. *numbers[3]* = 4 corresponds to Roman numeral 'IV'.
4. *numbers[4]* = 5 corresponds to Roman numeral 'V'.

Return the array *["I", "II", "III", "IV", "V"]* as the answer.

▼ Sample Case 1

Sample Input 1

```
STDIN    Function
-------   -----------
5     →   numbers[] size n = 5
75    →   numbers = [75, 80, 99, 100, 50]
80
99
100
50
```

Sample Output 1

```
LXXV
LXXX
XCIX
C
L
```

Explanation 1

Perform the following conversions:

0. *numbers[0]* = 75 corresponds to Roman numeral 'LXXV', 'L' (50) + 'X' (10) + 'X' (10) + 'V' (5).
1. *numbers[1]* = 80 corresponds to Roman numeral 'LXXX, 'L' (50) + 'X' (10) + 'X' (10) + 'X' (10).
2. *numbers[2]* = 99 corresponds to Roman numeral 'XCIX', 'XC' (90) + 'IX' (9).
3. *numbers[3]* = 100 corresponds to Roman numeral 'C'.
4. *numbers[4]* = 50 corresponds to Roman numeral 'L'.

Return the array ["LXXV", "LXXX", "XCIX", "C", "L"] as the answer.