Consider an array of numeric strings where each string is a positive number with anywhere from 1 to 10^6 digits. Sort the array's elements in non-decreasing, or ascending order of their integer values and return the sorted array.

Example

$unsorted = [\text{'1'}, \text{'200'}, \text{'150'}, \text{'3'}]$

Return the array ['1', '3', '150', '200'].

Function Description

Complete the bigSorting function in the editor below.

bigSorting has the following parameter(s):

- string unsorted[n]: an unsorted array of integers as strings

Returns

- string[n]: the array sorted in numerical order

Input Format

The first line contains an integer, n, the number of strings in $unsorted$.

Each of the n subsequent lines contains an integer string, $unsorted[i]$.

Constraints

- $1 \leq n \leq 2 \times 10^5$
- Each string is guaranteed to represent a positive integer.
- There will be no leading zeros.
- The total number of digits across all strings in $unsorted$ is between 1 and 10^6 (inclusive).

Sample Input 0

Change Theme Language C

```c
/*
aditya
1DT23CA008
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);

/*
 * Complete the 'bigSorting' function below.
 *
 * The function is expected to return a STRING_ARRAY.
 * The function accepts STRING_ARRAY unsorted as parameter.
```

Line: 8 Col: 18

Upload Code as File Test against custom input Run Code Submit Code

The previous challenges covered Insertion Sort, which is a simple and intuitive sorting algorithm with a running time of $O(n^2)$. In these next few challenges, we're covering a divide-and-conquer algorithm called Quicksort (also known as Partition Sort). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

Step 1: Divide

Choose some pivot element, p, and partition your unsorted array, arr, into three smaller arrays: $left$, $right$, and $equal$, where each element in $left < p$, each element in $right > p$, and each element in $equal = p$.

Example

$arr = [5, 7, 4, 3, 8]$

In this challenge, the pivot will always be at $arr[0]$, so the pivot is 5.

arr is divided into $left = \{4, 3\}$, $equal = \{5\}$, and $right = \{7, 8\}$. Putting them all together, you get $\{4, 3, 5, 7, 8\}$. There is a flexible checker that allows the elements of $left$ and $right$ to be in any order. For example, $\{3, 4, 5, 8, 7\}$ is valid as well.

Given arr and $p = arr[0]$, partition arr into $left$, $right$, and $equal$ using the Divide instructions above. Return a 1-dimensional array containing each element in $left$ first, followed by each element in $equal$, followed by each element in $right$.

Function Description

Complete the quickSort function in the editor below.

quickSort has the following parameter(s):

- int arr[n]: $arr[0]$ is the pivot element

Change Theme Language C

```c
/*
aditya
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'quickSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
```

Line: 4 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Comparison Sorting

Quicksort usually has a running time of $n \times log(n)$, but is there an algorithm that can sort even faster? In general, this is not possible. Most sorting algorithms are comparison sorts, i.e. they sort a list just by comparing the elements to one another. A comparison sort algorithm cannot beat $n \times log(n)$ (worst-case) running time, since $n \times log(n)$ represents the minimum number of comparisons needed to know where to place each element. For more details, you can see these notes (PDF).

Alternative Sorting

Another sorting method, the counting sort, does not require comparison. Instead, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.

Example

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

All of the values are in the range $[0 \ldots 3]$, so create an array of zeros, $result = [0, 0, 0, 0]$. The results of each iteration follow:

i	arr[i]	result
0	1	[0, 1, 0, 0]
1	1	[0, 2, 0, 0]
2	3	[0, 2, 0, 1]
3	2	[0, 2, 1, 1]
4	1	[0, 3, 1, 1]

The frequency array is $[0, 3, 1, 1]$. These values can be used to create the sorted array

Change Theme Language C

```c
/*
aditya
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'countingSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
```

Line: 4 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Consider an array of numeric strings where each string is a positive number with anywhere from 1 to 10^6 digits. Sort the array's elements in non-decreasing, or ascending order of their integer values and return the sorted array.

Example

$unsorted = [\text{'1'}, \text{'200'}, \text{'150'}, \text{'3'}]$

Return the array ['1', '3', '150', '200'].

Function Description

Complete the bigSorting function in the editor below.

bigSorting has the following parameter(s):

- string unsorted[n]: an unsorted array of integers as strings

Returns

- string[n]: the array sorted in numerical order

Input Format

The first line contains an integer, n, the number of strings in $unsorted$.

Each of the n subsequent lines contains an integer string, $unsorted[i]$.

Constraints

- $1 \leq n \leq 2 \times 10^5$
- Each string is guaranteed to represent a positive integer.
- There will be no leading zeros.
- The total number of digits across all strings in $unsorted$ is between 1 and 10^6 (inclusive).

Sample Input 0

Change Theme Language C

```c
/*
aditya
1DT23CA008
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);

/*
 * Complete the 'bigSorting' function below.
 *
 * The function is expected to return a STRING_ARRAY.
 * The function accepts STRING_ARRAY unsorted as parameter.
```

Line: 196 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Sorting

One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.

Insertion Sort

These challenges will cover Insertion Sort, a simple and intuitive sorting algorithm. We will first start with a nearly sorted list.

Insert element into sorted list

Given a sorted list with an unsorted number e in the rightmost cell, can you write some simple code to insert e into the array so that it remains sorted?

Since this is a learning exercise, it won't be the most efficient way of performing the insertion. It will instead demonstrate the brute-force method in detail.

Assume you are given the array $arr = [1, 2, 4, 5, 3]$ indexed $0 \ldots 4$. Store the value of $arr[4]$. Now test lower index values successively from 3 to 0 until you reach a value that is lower than $arr[4]$, at $arr[1]$ in this case. Each time your test fails, copy the value at the lower index to the current index and print your array. When the next lower indexed value is smaller than $arr[4]$, insert the stored value at the current index and print the entire array.

Example

$n = 5$

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

Start at the rightmost index. Store the value of $arr[4] = 3$. Compare this to each element to the left until a smaller value is reached. Here are the results as described:

Change Theme Language C

```c
/*
aditya
1DT23CA008
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'insertionSort1' function below.
 *
 * The function accepts following parameters:
```

Line: 173 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Problem

Submissions

Leaderboard

Discussions

In Insertion Sort Part 1, you inserted one element into an array at its correct sorted position. Using the same approach repeatedly, can you sort an entire array?

Guideline: You already can place an element into a sorted array. How can you use that code to build up a sorted array, one element at a time? Note that in the first step, when you consider an array with just the first element, it is already sorted since there's nothing to compare it to.

In this challenge, print the array after each iteration of the insertion sort, i.e., whenever the next element has been inserted at its correct position. Since the array composed of just the first element is already sorted, begin printing after placing the second element.

Example.

$n = 7$

$arr = [3, 4, 7, 5, 6, 2, 1]$

Working from left to right, we get the following output:

```
3 4 7 5 6 2 1
3 4 7 5 6 2 1
3 4 5 7 6 2 1
3 4 5 6 7 2 1
2 3 4 5 6 7 1
1 2 3 4 5 6 7
```

Function Description

Complete the insertionSort2 function in the editor below.

insertionSort2 has the following parameter(s):

Change Theme Language C

```c
/*
aditya
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'insertionSort2' function below.
 *
 * The function accepts following parameters:
 *  1. INTEGER n
```

Line: 172 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

The previous challenges covered Insertion Sort, which is a simple and intuitive sorting algorithm with a running time of $O(n^2)$. In these next few challenges, we're covering a divide-and-conquer algorithm called Quicksort (also known as Partition Sort). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

Step 1: Divide

Choose some pivot element, p, and partition your unsorted array, arr, into three smaller arrays: $left$, $right$, and $equal$, where each element in $left < p$, each element in $right > p$, and each element in $equal = p$.

Example

$arr = [5, 7, 4, 3, 8]$

In this challenge, the pivot will always be at $arr[0]$, so the pivot is 5.

arr is divided into $left = \{4, 3\}$, $equal = \{5\}$, and $right = \{7, 8\}$. Putting them all together, you get $\{4, 3, 5, 7, 8\}$. There is a flexible checker that allows the elements of $left$ and $right$ to be in any order. For example, $\{3, 4, 5, 8, 7\}$ is valid as well.

Given arr and $p = arr[0]$, partition arr into $left$, $right$, and $equal$ using the Divide instructions above. Return a 1-dimensional array containing each element in $left$ first, followed by each element in $equal$, followed by each element in $right$.

Function Description

Complete the quickSort function in the editor below.

quickSort has the following parameter(s):

- int arr[n]: $arr[0]$ is the pivot element

Change Theme Language C

```c
/*
aditya
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'quickSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
```

Line: 213 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Comparison Sorting

Quicksort usually has a running time of $n \times log(n)$, but is there an algorithm that can sort even faster? In general, this is not possible. Most sorting algorithms are comparison sorts, i.e. they sort a list just by comparing the elements to one another. A comparison sort algorithm cannot beat $n \times log(n)$ (worst-case) running time, since $n \times log(n)$ represents the minimum number of comparisons needed to know where to place each element. For more details, you can see these notes (PDF).

Alternative Sorting

Another sorting method, the counting sort, does not require comparison. Instead, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.

Example

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

All of the values are in the range $[0 \ldots 3]$, so create an array of zeros, $result = [0, 0, 0, 0]$. The results of each iteration follow:

i	arr[i]	result
0	1	[0, 1, 0, 0]
1	1	[0, 2, 0, 0]
2	3	[0, 2, 0, 1]
3	2	[0, 2, 1, 1]
4	1	[0, 3, 1, 1]

The frequency array is $[0, 3, 1, 1]$. These values can be used to create the sorted array

Change Theme Language C

```c
/*
aditya
*/
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'countingSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
```

stdlib.h
/usr/include/stdlib.h
provides EXIT_FAILURE , exit , getenv , malloc , realloc and 1 more

Line: 213 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Use the counting sort to order a list of strings associated with integers. If two strings are associated with the same integer, they must be printed in their original order, i.e. your sorting algorithm should be stable. There is one other twist: strings in the first half of the array are to be replaced with the character - (dash, ascii 45 decimal).

Insertion Sort and the simple version of Quicksort are stable, but the faster in-place version of Quicksort is not since it scrambles around elements while sorting.

Design your counting sort to be stable.

Example

$arr = [[0,'a'],[1,'b'],[0,'c'],[1,'d']]$

The first two strings are replaced with '-'. Since the maximum associated integer is 1, set up a helper array with at least two empty arrays as elements. The following shows the insertions into an array of three empty arrays.

i	string	converted list	
0			[[],[],[]]
1	a	-	[[-],[],[]]
2	b	-	[[-],[-],[]]
3	c		[[-,c],[-],[]]
4	d		[[-,c],[-,d],[]]

The result is then printed: **- c - d** .

Function Description

Complete the countSort function in the editor below. It should construct and print the sorted strings.

countSort has the following parameter(s):

Change Theme Language C

```c
/*
aditya
1DT23CA008
/*
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'countSort' function below.
 *
 * The function accepts 2D_STRING_ARRAY arr as parameter.
```

Line: 5 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Given an amount and the denominations of coins available, determine how many ways change can be made for amount. There is a limitless supply of each coin type.

Example

$n = 3$

$c = [8, 3, 1, 2]$

There are 3 ways to make change for $n = 3$: $\{1, 1, 1\}$, $\{1, 2\}$, and $\{3\}$.

Function Description

Complete the getWays function in the editor below.

getWays has the following parameter(s):

- int n: the amount to make change for
- int c[m]: the available coin denominations

Returns

- int: the number of ways to make change

Input Format

The first line contains two space-separated integers n and m, where:

n is the amount to change

m is the number of coin types

The second line contains m space-separated integers that describe the values of each coin type.

Constraints

- $1 \leq c[i] \leq 50$

Change Theme Language C

```
1  /*
2  aditya
3  1DT23CA008
4  */
5  #include <assert.h>
6  #include <ctype.h>
7  #include <limits.h>
8  #include <math.h>
9  #include <stdbool.h>
10 #include <stddef.h>
11 #include <stdint.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15
16 char* readline();
17 char* ltrim(char*);
18 char* rtrim(char*);
19 char** split_string(char*);
20
21 int parse_int(char*);
22 long parse_long(char*);
23
24 /*
25  * Complete the 'getWays' function below.
26  *
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

Implement a modified Fibonacci sequence using the following definition:

> Given terms $t[i]$ and $t[i+1]$ where $i \in (1, \infty)$, term $t[i+2]$ is computed as:
>
> $$t_{i+2} = t_i + (t_{i+1})^2$$

Given three integers, $t1$, $t2$, and n, compute and print the n^{th} term of a modified Fibonacci sequence.

Example

$t1 = 0$

$t2 = 1$

$n = 6$

- $t3 = 0 + 1^2 = 1$
- $t4 = 1 + 1^2 = 2$
- $t5 = 1 + 2^2 = 5$
- $t6 = 2 + 5^2 = 27$

Return 27.

Function Description

Complete the fibonacciModified function in the editor below. It must return the n^{th} number in the sequence.

fibonacciModified has the following parameter(s):

- int t1: an integer
- int t2: an integer
- int n: the iteration to report

Change Theme Language C

```c
/*
aditya
1DT23CA008
*/

#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'fibonacciModified' function below.
 *
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

Penny has an array of n integers, $[a_0, a_1, \ldots, a_{n-1}]$. She wants to find the number of unique multisets she can form using elements from the array such that the bitwise XOR of all the elements of the multiset is a prime number. Recall that a multiset is a set which can contain duplicate elements.

Given q queries where each query consists of an array of integers, can you help Penny find and print the number of valid multisets for each array? As these values can be quite large, modulo each answer by $10^9 + 7$ before printing it on a new line.

Input Format

The first line contains a single integer, q, denoting the number of queries. The $2 \cdot q$ subsequent lines describe each query in the following format:

1. The first line contains a single integer, n, denoting the number of integers in the array.
2. The second line contains n space-separated integers describing the respective values of $a_0, a_1, \ldots, a_{n-1}$.

Constraints

- $1 \leq q \leq 10$
- $1 \leq n \leq 100000$
- $3500 \leq a_i \leq 4500$

Output Format

On a new line for each query, print a single integer denoting the number of unique multisets Penny can construct using numbers from the array such that the bitwise

Change Theme Language C

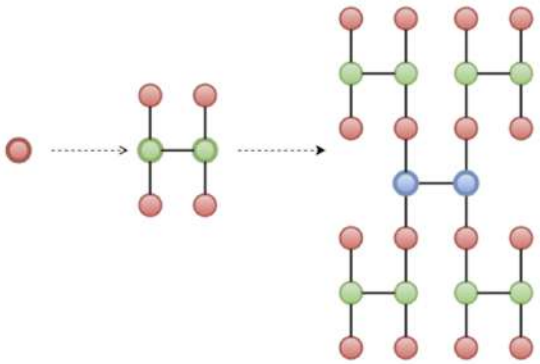
```c
/*
ADITYA
1DT23CA008
*/

#include <stdio.h>

#define MOD 1000000007
#define MAX_XOR 8192
#define MIN_VALUE 3500
#define MAX_VALUE 4500

int primeXor(int arr[], int n) {
    int valueCount[MAX_VALUE - MIN_VALUE + 1];
    int xorCount[MAX_XOR];
    int delta[MAX_XOR];
    int result = 0;

    // Initialize arrays
    for (int i = 0; i < MAX_VALUE - MIN_VALUE + 1; i++) {
        valueCount[i] = 0;
    }
    for (int i = 0; i < MAX_XOR; i++) {
        xorCount[i] = 0;
    }
    for (int i = 0; i < MAX_XOR; i++) {
```

Line: 4 Col: 3

Upload Code as File | Test against custom input | Run Code | Submit Code

HackerRank-city is an acyclic connected graph (or tree). Its not an ordinary place, the construction of the whole tree takes place in N steps. The process is described below:

- It initially has 1 node.
- At each step, you must create 3 duplicates of the current tree, and create 2 new nodes to connect all 4 copies in the following H shape:



At each i^{th} step, the tree becomes 4 times bigger plus 2 new nodes, as well as 5 new edges connecting everything together. The length of the new edges being added at step i is denoted by input A_i.

Calculate the sum of distances between each pair of nodes; as these answers may run large, print your answer modulo 1000000007.

Change Theme Language C

```c
/*
ADITYA
1DT23CA008
*/

#include <stdio.h>

#define MOD 1000000007

int main() {
    int n;
    scanf("%d", &n);
    long long nv = 1;    // Number of vertices
    long long sd = 0;    // Sum of distances
    long long dtc = 0;   // Distance to center
    long long diam = 0;  // Diameter

    for (int i = 1; i <= n; ++i) {
        int ne;
        scanf("%d", &ne);
        long long nnv = nv * 4 + 2;
        long long nsd = (4 * sd) % MOD;
        nsd = (nsd + (4 * dtc % MOD) * ((3 * nv + 2) % MOD)) % MOD;
        nsd = (nsd + (4 * ne % MOD) * nv % MOD * ((3 * nv + 2) % MOD)) % MO
        nsd = (nsd + ne % MOD * ((2 * nv + 1) % MOD) * ((2 * nv + 1) % MOD)
        % MOD;
```

Line: 4 Col: 3

Upload Code as File | Test against custom input | Run Code | Submit Code

Given an array of integers and a target sum, determine the sum nearest to but not exceeding the target that can be created. To create the sum, use any element of your array zero or more times.

For example, if $arr = [2, 3, 4]$ and your target sum is 10, you might select $[2, 2, 2, 2, 2]$, $[2, 2, 3, 3]$ or $[3, 3, 3, 1]$. In this case, you can arrive at exactly the target.

Function Description

Complete the unboundedKnapsack function in the editor below. It must return an integer that represents the sum nearest to without exceeding the target value.

unboundedKnapsack has the following parameter(s):

- k: an integer
- arr: an array of integers

Input Format

The first line contains an integer t, the number of test cases.

Each of the next t pairs of lines are as follows:

- The first line contains two integers n and k, the length of arr and the target sum.
- The second line contains n space separated integers $arr[i]$.

Constraints

$1 \leq t \leq 10$

$1 \leq n, k, arr[i] \leq 2000$

Output Format

Print the maximum sum for each test case which is as near as possible, but not

Change Theme Language C

```
1  /*
2  ADITYA
3  1DT23CA008
4  */
5
6  #include <assert.h>
7  #include <ctype.h>
8  #include <limits.h>
9  #include <math.h>
10 #include <stdbool.h>
11 #include <stddef.h>
12 #include <stdint.h>
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16
17 char* readline();
18 char* ltrim(char*);
19 char* rtrim(char*);
20 char** split_string(char*);
21
22 int parse_int(char*);
23
24 /*
25  * Complete the 'unboundedKnapsack' function below.
26  *
```

Line: 4 Col: 3

Upload Code as File | Test against custom input | Run Code | Submit Code

There are N cities and N directed roads in Steven's world. The cities are numbered from 0 to N - 1. Steven can travel from city i to city (i + 1) % N. (0-> 1 -> 2 -> .... -> N - 1 -> 0).

Steven wants to travel around the world by car. The capacity of his car's fuel tank is C gallons. There are a[i] gallons he can use at the beginning of city i and the car takes b[i] gallons to travel from city i to (i + 1) % N.

How many cities can Steven start his car from so that he can travel around the world and reach the same city he started?

Note

The fuel tank is initially empty.

Input Format

The first line contains two integers (separated by a space): city number N and capacity C.
The second line contains N space-separated integers: a[0], a[1], ... , a[N - 1].
The third line contains N space-separated integers: b[0], b[1], ... , b[N - 1].

Constraints

$2 \le N \le 10^5$
$1 \le C \le 10^{18}$
$0 \le a[i], b[i] \le 10^9$

Output Format

The number of cities which can be chosen as the start city.

Sample Input

Change Theme Language C

```c
/*
ADITYA
1DT23CA008
*/

#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);
long parse_long(char*);

/*
 * Complete the 'travelAroundTheWorld' function below.
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

There are two n-element arrays of integers, A and B. Permute them into some A' and B' such that the relation $A'[i] + B'[i] \geq k$ holds for all i where $0 \leq i < n$.

There will be q queries consisting of A, B, and k. For each query, return YES if some permutation A', B' satisfying the relation exists. Otherwise, return NO.

Example

$A = [0, 1]$

$B = [0, 2]$

$k = 1$

A valid A', B' is $A' = [1, 0]$ and $B' = [0, 2]$: $1 + 0 \geq 1$ and $0 + 2 \geq 1$. Return YES.

Function Description

Complete the twoArrays function in the editor below. It should return a string, either YES or NO.

twoArrays has the following parameter(s):

- int k: an integer
- int A[n]: an array of integers
- int B[n]: an array of integers

Returns

- string: either YES or NO

Input Format

The first line contains an integer q, the number of queries.

The next q sets of 3 lines are as follows:

- The first line contains two space-separated integers n and k, the size of both

Change Theme Language C

```c
/*
ADITYA
1DT23CA008
*/


#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'twoArrays' function below.
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

Jenna is playing a computer game involving a large map with n cities numbered sequentially from 1 to n that are connected by m bidirectional roads. The game's objective is to travel to as many cities as possible without visiting any city more than once. The more cities the player visits, the more points they earn.

As Jenna's fellow student at Hackerland University, she asks you for help choosing an optimal path. Given the map, can you help her find a path that maximizes her score?

Note: She can start and end her path at any two distinct cities.

Input Format

The first line contains two space-separated integers describing the respective values of n (the number of cities) and m (the number of roads).
Each line i of the m subsequent lines contains two space-separated integers, x_i and y_i, describing a bidirectional road between cities x_i and y_i.

Map Generation Algorithm

The graph representing the map was generated randomly in the following way:

1. Initially, the graph was empty.
2. Permutations $p_1, \ldots, p_n$ were chosen uniformly at random among all $n!$ permutations.
3. For each $i \in \{1, \ldots, n-1\}$, edge (p_i, p_{i+1}) was added to the graph.
4. An additional $m - n + 1$ edges were chosen uniformly at random among all possible sets of $m - n + 1$ edges which don't intersect with edges added during step 3.

Constraints

Change Theme Language C

```c
/*
ADITYA
1DT23CA008
*/
#include <stdio.h>
#include <stdlib.h>

#define MAXN 100005

int n, m;
int *adj[MAXN];
int deg[MAXN];
int visited[MAXN];
int path[MAXN];
int path_len = 0;

// Function to compare degrees for qsort
int cmp(const void *a, const void *b) {
    int u = *(int *)a;
    int v = *(int *)b;
    return deg[u] - deg[v];
}

void dfs(int u) {
    visited[u] = 1;
    path[path_len++] = u;
```

Line: 4 Col: 3

Upload Code as File | Test against custom input | Run Code | Submit Code

A regular expression is used to describe a set of strings. For this problem the alphabet is limited to 'a' and 'b'.

We define R to be a valid regular expression if:

1) R is "a" or "b".

2) R is of the form "$(R_1 R_2)$", where R_1 and R_2 are regular expressions.

3) R is of the form "$(R_1|R_2)$" where R_1 and R_2 are regular expressions.

4) R is of the form "(R_1*)" where R_1 is a regular expression.

Regular expressions can be nested and will always have have two elements in the parentheses. ('*' is an element, '|' is not; basically, there will always be pairwise evaluation) Additionally, '*' will always be the second element; '$(*a)$' is invalid.

The set of strings recognized by R are as follows:

1) If R is "a", then the set of strings recognized $= a$.

2) If R is "b", then the set of strings recognized $= b$.

3) If R is of the form "$(R_1 R_2)$" then the set of strings recognized = all strings which can be obtained by a concatenation of strings s_1 and s_2, where s_1 is recognized by R_1 and s_2 by R_2.

4) If R is of the form "$(R1|R2)$" then the set of strings recognized = union of the set of strings recognized by R_1 and R_2.

5) If R is of the form "(R_1*)" then the the strings recognized are the empty string and the concatenation of an arbitrary number of copies of any string recognized by R_1.

Task

Given a regular expression and an integer, L, count how many strings of length L are recognized by it.

Change Theme Language Python 3

```
/*
ADITYA
1DT23CA008
*/

import sys
import threading

MOD = 10 ** 9 + 7

def main():
    import re
    sys.setrecursionlimit(100000)

    class State:
        def __init__(self):
            self.edges = {}  # char -> set of State
            self.epsilon = set()
            self.id = None
            self.accept = False

    def parse_regex(expr):
        """Parse the regex using Thompson's construction."""
        def precedence(op):
            if op == '*':
                return 3
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

Jane loves strings more than anything. She has a string t with her, and value of string s over function f can be calculated as given below:

$$f(s) = |s| \times \text{Number of times } s \text{ occurs in } t$$

Jane wants to know the maximum value of $f(s)$ among all the substrings (s) of string t. Can you help her?

Input Format

A single line containing string t .

Output Format

Print the maximum value of $f(s)$ among all the substrings (s) of string t.

Constraints

$1 \leq |t| \leq 10^5$

The string consists of lowercase English alphabets.

Sample Input 0

```
aaaaaa
```

Sample Output 0

```
12
```

Explanation 0

```
f('a') = 6
f('aa') = 10
```

Change Theme Language C

```c
/*
ADITYA
1DT23CA008
*/

#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

/*
 * Complete the 'maxValue' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING t as parameter.
 */

int maxValue(char* t) {
```

Line: 4 Col: 3

Upload Code as File Test against custom input Run Code Submit Code

All Submissions

Submissions

Problem	Language	Time	Result	Score	Action
Count Strings	c	about 1 hour	Compilation error	0.0	View results
Count Strings	python3	about 1 hour	Accepted	80.0	View results
Count Strings	java	about 1 hour	Compilation error	0.0	View results
Count Strings	cpp	about 1 hour	Compilation error	0.0	View results
Count Strings	c	about 1 hour	Compilation error	0.0	View results
Walking the Approximate Longest Path	python3	about 1 hour	Processed	0.0	View results
Walking the Approximate Longest Path	c	about 1 hour	Accepted	43.75	View results
Permuting Two Arrays	c	about 1 hour	Compilation error	0.0	View results
Fibonacci Modified	c	about 2 hours	Compilation error	0.0	View results
Knapsack	java	about 2 hours	Compilation error	0.0	View results