1.

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:

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

Function Description

Complete the *insertionSort1* function in the editor below.

insertionSort1 has the following parameter(s):

- *n:* an integer, the size of arr
- *arr:* an array of integers to sort

Returns

- *None:* Print the interim and final arrays, each on a new line. No return value is expected.

Input Format

The first line contains the integer n, the size of the array arr.
The next line contains n space-separated integers $arr[0] \ldots arr[n-1]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000$

Output Format

Print the array as a row of space-separated integers each time there is a shift or insertion.

2.

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):

- *int n:* the length of arr
- *int arr[n]:* an array of integers

Prints

At each iteration, print the array as space-separated integers on its own line.

Input Format

The first line contains an integer, n, the size of arr.
The next line contains n space-separated integers $arr[i]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000, 0 \leq i < n$

Output Format

Print the entire array on a new line at every iteration.

3.

Starting with a 1-indexed array of zeros and a list of operations, for each operation add a value to each the array element between two given indices, inclusive. Once all operations have been performed, return the maximum value in the array.

Example

$n = 10$
$queries = [[1, 5, 3], [4, 8, 7], [6, 9, 1]$

Queries are interpreted as follows:

```
a b k
1 5 3
4 8 7
6 9 1
```

Add the values of k between the indices a and b inclusive:

```
index->  1 2 3  4  5 6 7 8 9 10
       [0,0,0, 0, 0,0,0,0,0, 0]
       [3,3,3, 3, 3,0,0,0,0, 0]
       [3,3,3,10,10,7,7,7,0, 0]
       [3,3,3,10,10,8,8,8,1, 0]
```

The largest value is 10 after all operations are performed.

Function Description

Complete the function *arrayManipulation* in the editor below.

arrayManipulation has the following parameters:

- *int n* - the number of elements in the array
- *int queries[q][3]* - a two dimensional array of queries where each *queries[i]* contains three integers, *a*, *b*, and *k*.

Returns

- *int* - the maximum value in the resultant array

Input Format

The first line contains two space-separated integers n and m, the size of the array and the number of operations.
Each of the next m lines contains three space-separated integers a, b and k, the left index, right index and summand.

Constraints

- $3 \leq n \leq 10^7$
- $1 \leq m \leq 2 * 10^5$
- $1 \leq a \leq b \leq n$
- $0 \leq k \leq 10^9$