

An array is a data structure that stores elements of the same type in a contiguous block of memory. In an array,  $A$ , of size  $N$ , each memory location has some unique index,  $i$  (where  $0 \leq i < N$ ), that can be referenced as  $A[i]$  or  $A_i$ .

Your task is to reverse an array of integers.

**Note:** If you've already solved our C++ domain's Arrays Introduction challenge, you may want to skip this.

#### Example

$A = [1, 2, 3]$

Return  $[3, 2, 1]$ .

#### Function Description

Complete the function *reverseArray* with the following parameter(s):

- $int\ A[n]$ : the array to reverse

#### Returns

- $int[n]$ : the reversed array

#### Input Format

## Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

### ✓ Sample Test case 0

Input (stdin)

[Download](#)

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

Your Output (stdout)

```
1 2 3 4 1
```

Expected Output

[Download](#)

```
1 2 3 4 1
```

Given a  $6 \times 6$  2D array, *arr*, an hourglass is a subset of values with indices falling in the following pattern:

```
a b c
  d
e f g
```

There are 16 hourglasses in a  $6 \times 6$  array. The *hourglass sum* is the sum of the values in an hourglass. Calculate the hourglass sum for every hourglass in *arr*, then print the *maximum* hourglass sum.

#### Example

*arr* =

```
-9 -9 -9 1 1 1
 0 -9 0 4 3 2
-9 -9 -9 1 2 3
 0 0 8 6 6 0
 0 0 0 -2 0 0
 0 0 1 2 4 0
```

The 16 hourglass sums are:

```
-62 -24 -10 12
```

## Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

### ✓ Sample Test case 0

### ✓ Sample Test case 1

### ✓ Sample Test case 2

Input (stdin)

```
1 1 1 1 0 0 0
2 0 1 0 0 0 0
3 1 1 1 0 0 0
4 0 0 2 4 4 0
5 0 0 0 2 0 0
6 0 0 1 2 4 0
```

Download

Your Output (stdout)

```
1 19
```

Expected Output

```
1 19
```

Download

- Declare a 2-dimensional array, *arr*, with *n* empty arrays, all zero-indexed.
- Declare an integer, *lastAnswer*, and initialize it to 0.

You need to process two types of queries:

1. Query: 1 *x y*

- Compute  $idx = (x \oplus lastAnswer)$ .
- Append the integer *y* to *arr[idx]*.

2. Query: 2 *x y*

- Compute  $idx = (x \oplus lastAnswer)$ .
- Set  $lastAnswer = arr[idx][y \% size(arr[idx])]$ .
- Store the new value of *lastAnswer* in an answers array.

#### Notes:

- $\oplus$  is the bitwise XOR operation, which corresponds to the  $\wedge$  operator in most languages. Learn more about it on [Wikipedia](https://en.cppreference.com/w/cpp/algorithm/bitwise_xor).
- $\%$  is the modulo operator.
- Finally,  $size(arr[idx])$  is the number of elements in *arr[idx]*.

#### Function Description

Complete the *dynamicArray* function with the following parameters:

- *int n*: the number of empty arrays to initialize in *arr*
- *int queries[q][3]*: 2-D array of integers

## Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

### ✓ Sample Test case 0

Input (stdin)

```
1 2 5
2 1 0 5
3 1 1 7
4 1 0 3
5 2 1 0
6 2 1 1
```

Download

Your Output (stdout)

```
1 7
2 3
```

Expected Output

Download



A *left rotation* operation on a circular array shifts each of the array's elements 1 unit to the left. The elements that fall off the left end reappear at the right end. Given an integer  $d$ , rotate the array that many steps to the left and return the result.

Example

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

After 2 rotations,  $arr' = [3, 4, 5, 1, 2]$ .

Function Description

Complete the `rotateLeft` function with the following parameters:

- `int d`: the amount to rotate by
- `int arr[n]`: the array to rotate

Returns

- `int[n]`: the rotated array

Input Format

The first line contains two space-separated integers that denote  $n$ , the number of integers, and  $d$ , the number of left rotations to perform.  
The second line contains  $n$  space-separated integers that describe  $arr[]$ .

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Input (stdin)

Download

1 5 4  
2 1 2 3 4 5

Your Output (stdout)

1 5 1 2 3 4

Expected Output

Download

1 5 1 2 3 4



Starting with a 1-indexed array of zeros and a list of operations, for each operation add a value to each 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:

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

## Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

### ✓ Sample Test case 0

Input (stdin)

[Download](#)

### ✓ Sample Test case 1

### ✓ Sample Test case 2

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

Your Output (stdout)

1	200
---	-----

Expected Output

1	200
---	-----

[Download](#)