Reverse Array Queries

For a given array of integers, perform operations on the array. Return the resulting array after all operations have been applied in the order given. Each operation contains two indices. Reverse the subarray between those zero-based indices, inclusive.

**Example**

*arr = [5, 3, 2, 1, 3]*

*operations = [[0, 1], [1, 3]]*

In the first operation, reverse the subarray from *arr[0]* through *arr[1]: arr' = [3, 5, 2, 1, 3]*

In the second operation, reverse the subarray from *arr'[1]* through *arr'[3]: arr'' = [3, 1, 2, 5, 3]*

All operations have been performed, so return the array *[3, 1, 2, 5, 3].*

**Function Description**

Complete the function *performOperations* in the editor below.

The function has the following parameter(s):

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

*int* *operations[q][2]*: a 2-dimensional array of starting and ending indices

Returns:

*int[n]:* the final array after all operations have been performed

**Constraints**

* 1 ≤ *n, q* ≤ 103
* 1 ≤ *arr[i]* ≤ 103
* 0 ≤ *operations[i][0] ≤ operations[i][1]* < *n*

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

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

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

The next line contains an integer, *q,* the size of *operations*.

In the second line, there is a single integer *2,* the number of columns in each *operations[i]*.

Each of the following *q* lines contains two space-separated integers, *operations[i].*

**Sample Case 0**

**Sample Input**

STDIN Function

-----     -----

3     →  arr[] size n = 3

1      →  arr = [1, 2, 3]

2

3

3     →  operations[] size q = 3

2     →  operations[q][] size columns = 2 (always = 2)

0 2    →  operations = [[0, 2], [1, 2], [0, 2]]

1 2

0 2

**Sample Output**

2

1

3

**Explanation**

The original array *arr = [1, 2, 3].*

Reverse *arr[0]* through *arr[2]: arr' = [3, 2, 1]*.

Reverse *arr'[1]* through *arr[2]: arr'' = [3, 1, 2]*.

Reverse *arr''[0]* through *arr''[2]: arr''' = [2, 1, 3].*

**Sample Case 1**

**Sample Input**

STDIN Function

-----     --------

4     →  arr[] size n = 4

5      →  arr = [5, 2, 5, 1]

2

5

1

2     →  operations[] size q = 2

2     →  operations[q][] size columns = 2

1 2    →  operations = [[1, 2], [1, 1]]

1 1

**Sample Output**

5

5

2

1

**Explanation**

The original array *arr = [5, 2, 5, 1].*

Reverse *arr[1]* through *arr[2]: arr' = [5, 5, 2, 1]*.

Reverse *arr'[1]* through *arr[1]: arr'' = [5, 5, 2, 1]*.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'performOperations' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY arr

\* 2. 2D\_INTEGER\_ARRAY operations

\*/

public static List<Integer> performOperations(List<Integer> arr, List<List<Integer>> operations) {

// Write your code here

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arrCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = IntStream.range(0, arrCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int operationsRows = Integer.parseInt(bufferedReader.readLine().trim());

int operationsColumns = Integer.parseInt(bufferedReader.readLine().trim());

List<List<Integer>> operations = new ArrayList<>();

IntStream.range(0, operationsRows).forEach(i -> {

try {

operations.add(

Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList())

);

} catch (IOException ex) {

throw new RuntimeException(ex);

}

});

List<Integer> result = Result.performOperations(arr, operations);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*