



Programación Orientada a Eventos: Estructuras de Datos

Carlos Felipe Montoya Rincon

carlos.felipe.montoya@correounivalle.edu.co

Junio 20





TABLE OF CONTENTS

01

Estructuras de
Datos

02

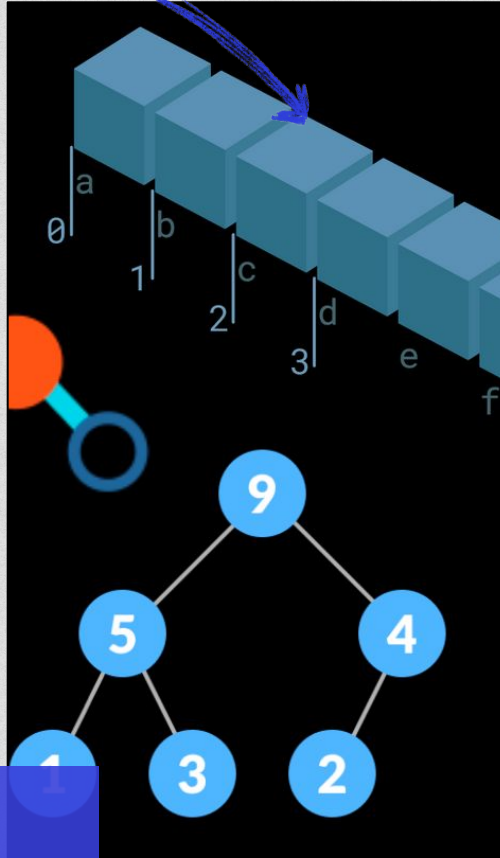
Diferencias entre
Estructuras de Datos

03

Estructuras de
Datos Lineales

04

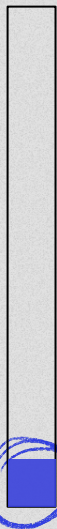
Estructuras de Datos
no Lineales



01

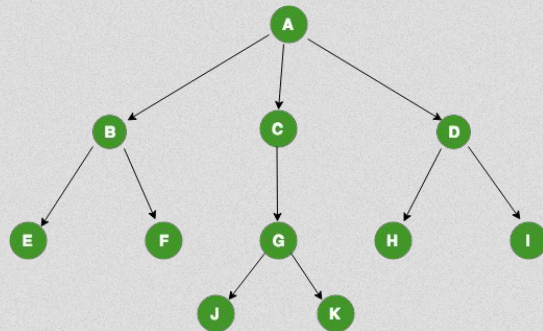
Estructuras de Datos

¿Qué son las estructuras de datos?



Las estructuras de datos son una forma de organizar y almacenar la información para ser utilizados de forma eficiente.

Proporcionan una manera de manejar grandes cantidades de datos de manera eficaz en términos de **tiempo** y **espacio**.



Importancia de las estructuras de datos



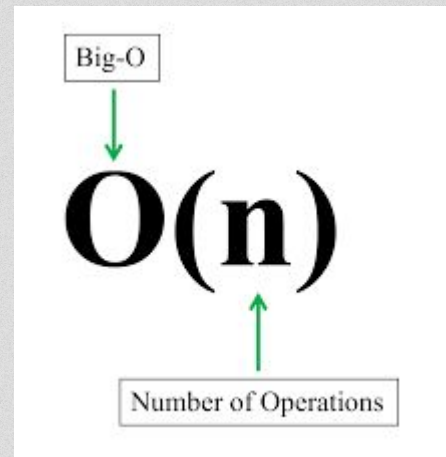
- Optimización del uso de recursos
- Organización de los datos
- Resolución de problemas
- Eficiencia en la codificación
- Diseño de algoritmos



Introducción Big O

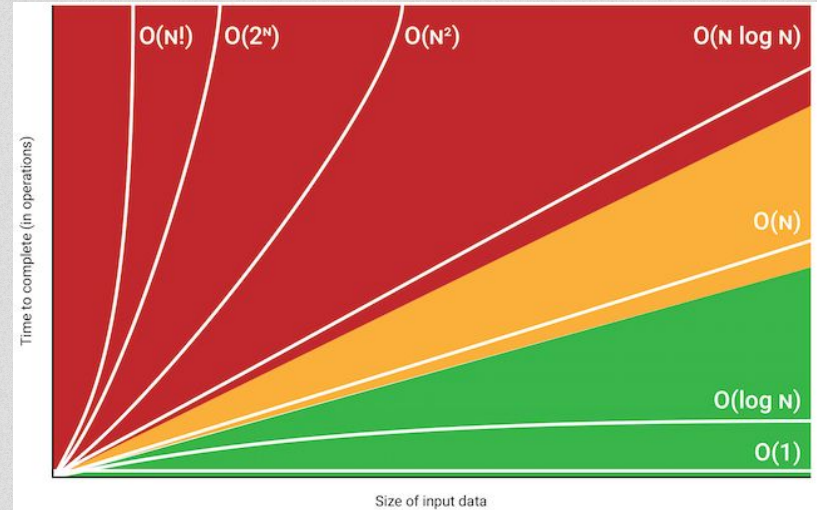


La notación Big O es una notación matemática la cual describe, el límite superior del tiempo de ejecución de un algoritmo en función del tamaño de la entrada.



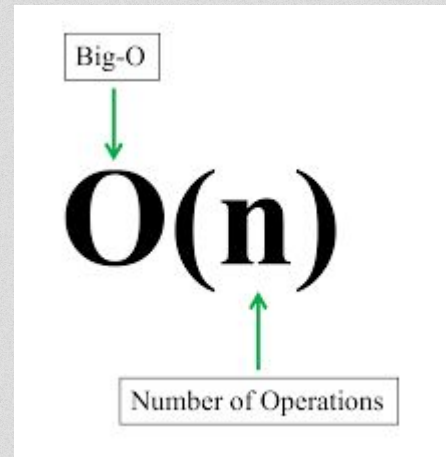
Introducción Big O

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$




Introducción Big O

Pero la notación Big O no solo sirve para medir el tiempo de un algoritmo, también para medir la complejidad espacial de un algoritmo.




Ejemplos de Complejidades



```
public void printFirstElement(int[] array) {  
    System.out.println(array[0]);  
}
```




Ejemplos de Complejidades



```
public void printAllElements(int[] array) {  
    for(int i = 0; i < array.length; i++) {  
        System.out.println(array[i]);  
    }  
}
```




Ejemplos de Complejidades



```
public void printAllPairs (int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        for (int j = 0; j < array.length; j++) {  
            System.out.println(array[i] + ", " + array[j]);  
        }  
    }  
}
```




Ejemplos de Complejidades



```
public void binarySearch(int[] sortedArray, int key) {  
    int low = 0;  
    int high = sortedArray.length - 1;  
    while(low <= high) {  
        int mid = low + (high - low) / 2;  
        if(key < sortedArray[mid]) {  
            high = mid - 1;  
        } else if(key > sortedArray[mid]) {  
            low = mid + 1;  
        } else {  
            System.out.println("Key found at index: " + mid);  
            return;  
        }  
    }  
    System.out.println("Key not found");  
}
```




¿Qué complejidades son?



```
public void printArrayBackwards(int[] array) {  
    for(int i = array.length - 1; i >= 0; i--) {  
        System.out.println(array[i]);  
    }  
}
```




¿Qué complejidades son?



```
public void mysteriousSort(int[] array) {  
    for(int i = 0; i < array.length - 1; i++) {  
        for(int j = 0; j < array.length - i - 1; j++) {  
            if(array[j] > array[j + 1]) {  
                int temp = array[j];  
                array[j] = array[j + 1];  
                array[j + 1] = temp;  
            }  
        }  
    }  
}
```



¿Qué complejidades son?



```
public boolean isEven(int number) {  
    return number % 2 == 0;  
}
```





Complejidades en Espacio



```
public int findMax(int[] array) {  
    int max = Integer.MIN_VALUE;  
    for(int i = 0; i < array.length; i++) {  
        if(array[i] > max) {  
            max = array[i];  
        }  
    }  
    return max;  
}
```




Complejidades en Espacio



```
public int[] duplicateArray(int[] array) {  
    int[] newArray = new int[array.length];  
    for(int i = 0; i < array.length; i++) {  
        newArray[i] = array[i];  
    }  
    return newArray;  
}
```




Complejidades en Espacio



```
public int[][] generateMatrix(int n) {  
    int[][] matrix = new int[n][n];  
    for(int i = 0; i < n; i++) {  
        for(int j = 0; j < n; j++) {  
            matrix[i][j] = i * j;  
        }  
    }  
    return matrix;  
}
```




¿Qué complejidad es?



```
public int fibonacci(int n) {  
    if (n <= 1) {  
        return n;  
    }  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```





¿Qué complejidad es?



```
public void mysteriousAlgorithm (int node, boolean[] visited,
List<Integer>[] graph) {
    visited[node] = true;
    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            mysteriousAlgorithm (neighbor, visited, graph);
        }
    }
}
```



¿Qué complejidad es?



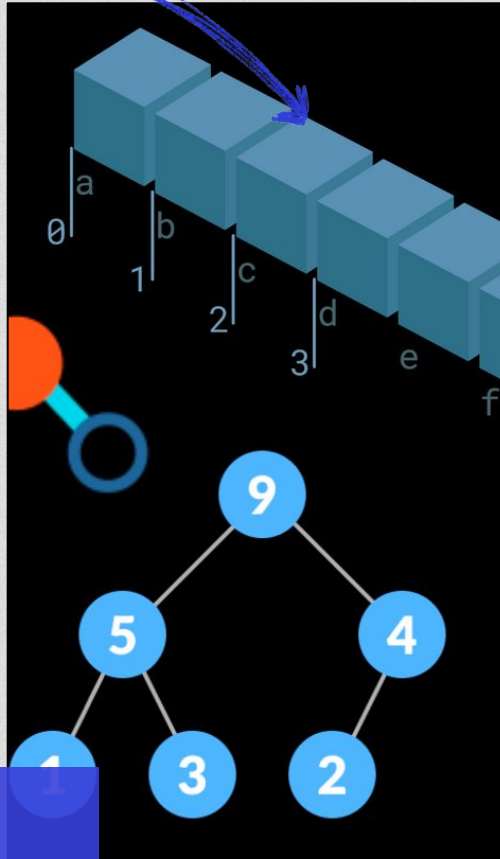
```
public void quicksort(int[] array, int low, int high) {  
    if(low < high) {  
        int pivot = partition(array, low, high);  
        quicksort(array, low, pivot-1);  
        quicksort(array, pivot+1, high);  
    }  
}
```



¿Qué complejidad es?



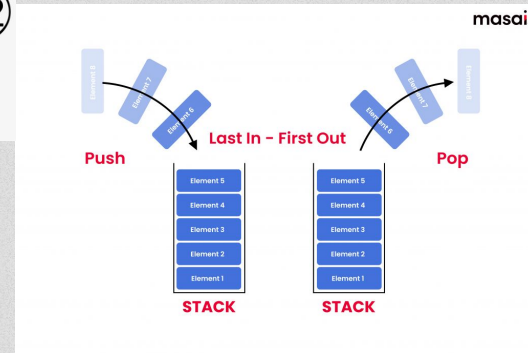
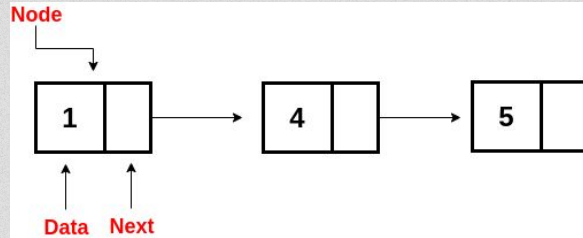
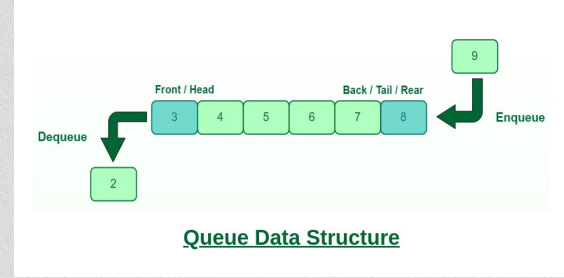
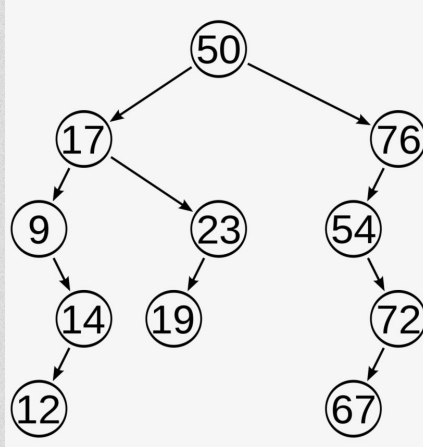
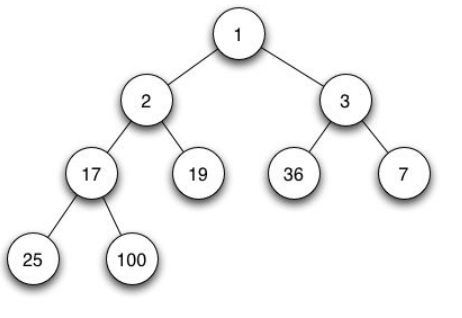
```
private int partition(int[] array, int low, int high) {  
    int pivot = array[high];  
    int i = low - 1;  
    for(int j = low; j < high; j++) {  
        if(array[j] < pivot) {  
            i++;  
            int temp = array[i];  
            array[i] = array[j];  
            array[j] = temp;  
        }  
    }  
    int temp = array[i+1];  
    array[i+1] = array[high];  
    array[high] = temp;  
    return i+1;  
}
```

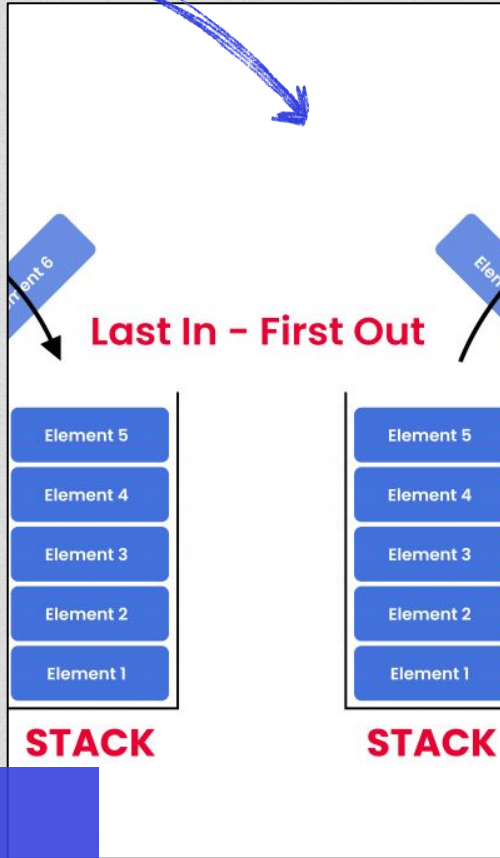



02

Diferencias entre Estructuras de Datos

¿Cuáles tipos de estructuras de datos hay?





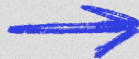
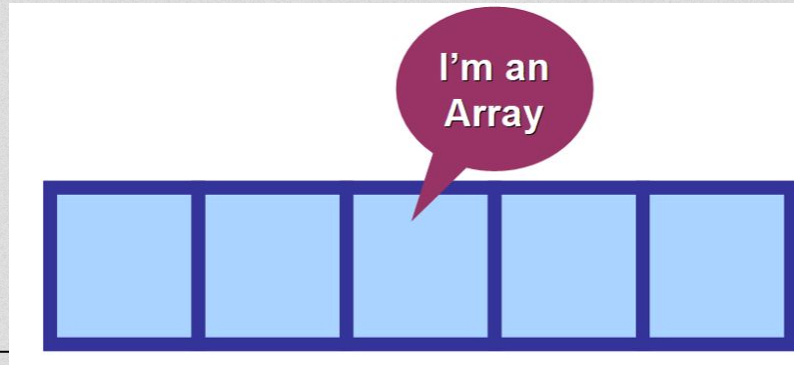
02

Estructuras de Datos Lineales


Arrays

Los arrays son estructuras de datos lineales que permiten almacenar información de forma continua. Algunas de sus ventajas son:

- Acceso de tiempo constante $O(1)$ a los elementos
- Uso eficiente de la memoria
- Facilita la manipulación de datos
- Expansión automática
- Adición y eliminación eficiente de elementos

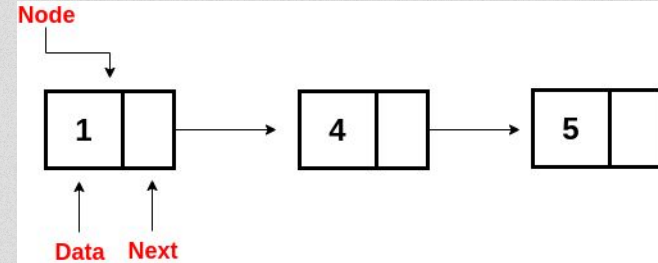


Linked List



La clase LinkedList en Java es una implementación de la estructura de datos de lista enlazada. Cada elemento en la lista enlazada es en realidad un nodo que tiene un valor y dos referencias, una al nodo anterior y otra al siguiente nodo. Esto es lo que permite la inserción y eliminación eficientes de elementos en cualquier parte de la lista. Ventajas:

- **Inserción y eliminación eficientes de elementos**
- **Inserción eficiente en ambos extremos**
- **Uso de memoria dinámico**



Linked List vs Array



Operación	ArrayList	LinkedList
Acceso por índice	$O(1)$	$O(n)$
Añadir al final	$O(1)$ amortizado	$O(1)$
Añadir en una posición específica	$O(n)$	$O(n)$
Añadir al principio	$O(n)$	$O(1)$
Eliminar al final	$O(1)$	$O(1)$
Eliminar en una posición específica	$O(n)$	$O(n)$
Eliminar al principio	$O(n)$	$O(1)$
Buscar un elemento específico	$O(n)$	$O(n)$



Ejercicios de Arrays o LinkedList



Amugae tiene un hotel compuesto por 10 habitaciones. Las habitaciones están numeradas del 0 al 9, de izquierda a derecha.

El hotel tiene dos entradas: una desde el extremo izquierdo y otra desde el extremo derecho. Cuando un cliente llega al hotel a través de la entrada izquierda, se le asigna la habitación vacía más cercana a la entrada izquierda. Del mismo modo, cuando un cliente llega al hotel a través de la entrada derecha, se le asigna la habitación vacía más cercana a la entrada derecha.

Un día, Amugae perdió la lista de asignación de habitaciones. Afortunadamente, la memoria de Amugae es perfecta y recuerda todos los clientes: cuándo llegó un cliente, desde qué entrada y cuándo salieron del hotel. Inicialmente, el hotel estaba vacío. Escribe un programa que recupere la lista de asignación de habitaciones a partir de los recuerdos de Amugae.



Ejercicios de Arrays o LinkedList

Entrada

La primera línea consta de un entero n ($1 \leq n \leq 105$), el número de eventos en la memoria de Amugae.

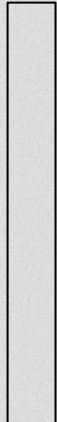
La segunda línea consta de una cadena de longitud n que describe los eventos en orden cronológico. Cada carácter representa:

- 'L': Un cliente llega desde la entrada izquierda.
- 'R': Un cliente llega desde la entrada derecha.
- '0', '1', ..., '9': El cliente de la habitación x (0, 1, ..., 9 respectivamente) se va.

Se garantiza que hay al menos una habitación vacía cuando llega un cliente, y hay un cliente en la habitación x cuando se da x (0, 1, ..., 9). Además, todas las habitaciones están inicialmente vacías.

Salida

En la única línea, muestra el estado de asignación de las habitaciones del hotel, desde la habitación 0 hasta la habitación 9. Representa una habitación vacía como '0' y una habitación ocupada como '1', sin espacios.



Ejercicios de Arrays o LinkedList

Examples

input

Copy

8
LLRL1RL1

output

Copy

1010000011

input

Copy

9
L0L0LLRR9

output

Copy

1100000010



Ejercicios de Arrays o LinkedList

En el primer ejemplo, el estado de asignación de las habitaciones del hotel después de cada acción es el siguiente:

- En primer lugar, todas las habitaciones están vacías. El estado de asignación es 0000000000.
- L: un cliente llega al hotel por la entrada izquierda. El estado de asignación es 1000000000.
- L: un cliente más llega por la entrada izquierda. El estado de asignación es 1100000000.
- R: un cliente más llega por la entrada derecha. El estado de asignación es 1100000001.
- L: un cliente más llega por la entrada izquierda. El estado de asignación es 1110000001.
- 1: el cliente en la habitación 1 se va. El estado de asignación es 1010000001.
- R: un cliente más llega por la entrada derecha. El estado de asignación es 1010000011.
- L: un cliente más llega por la entrada izquierda. El estado de asignación es 1110000011.
- 1: el cliente en la habitación 1 se va. El estado de asignación es 1010000011.

Por lo tanto, después de todo, el estado final de asignación de las habitaciones del hotel es 1010000011.



Ejercicios de Arrays o LinkedList

En el segundo ejemplo, el estado de asignación de las habitaciones del hotel después de cada acción es el siguiente:

- L: un cliente llega al hotel por la entrada izquierda. El estado de asignación es 1000000000.
- 0: el cliente en la habitación 0 se va. El estado de asignación es 0000000000.
- L: un cliente llega al hotel por la entrada izquierda. El estado de asignación vuelve a ser 1000000000.
- 0: el cliente en la habitación 0 se va. El estado de asignación es 0000000000.
- L: un cliente llega al hotel por la entrada izquierda. El estado de asignación es 1000000000.
- L: un cliente más llega por la entrada izquierda. El estado de asignación es 1100000000.
- R: un cliente más llega por la entrada derecha. El estado de asignación es 1100000001.
- R: un cliente más llega por la entrada derecha. El estado de asignación es 1100000011.
- 9: el cliente en la habitación 9 se va. El estado de asignación es 1100000010.

Por lo tanto, después de todo, el estado final de asignación de las habitaciones del hotel es 1100000010.



Ejercicios de Arrays o LinkedList



Given a **zero-based permutation** `nums` (**0-indexed**), build an array `ans` of the **same length** where `ans[i] = nums[nums[i]]` for each $0 \leq i < \text{nums.length}$ and return it.

A **zero-based permutation** `nums` is an array of **distinct** integers from `0` to `nums.length - 1` (**inclusive**).



Ejercicios de Arrays o LinkedList

Example 1:

Input: `nums = [0,2,1,5,3,4]`

Output: `[0,1,2,4,5,3]`

Explanation: The array `ans` is built as follows:

```
ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]  
      = [nums[0], nums[2], nums[1], nums[5], nums[3], nums[4]]  
      = [0,1,2,4,5,3]
```

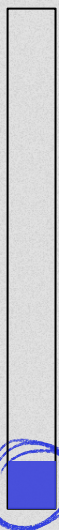
Example 2:

Input: `nums = [5,0,1,2,3,4]`

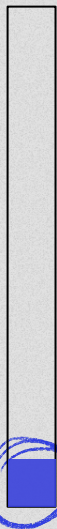
Output: `[4,5,0,1,2,3]`

Explanation: The array `ans` is built as follows:

```
ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]  
      = [nums[5], nums[0], nums[1], nums[2], nums[3], nums[4]]  
      = [4,5,0,1,2,3]
```



Ejercicios de Arrays o LinkedList



Given two arrays of integers `nums` and `index`. Your task is to create *target* array under the following rules:

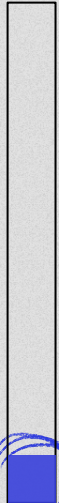
- Initially *target* array is empty.
- From left to right read `nums[i]` and `index[i]`, insert at index `index[i]` the value `nums[i]` in *target* array.
- Repeat the previous step until there are no elements to read in `nums` and `index`.

Return the *target* array.

It is guaranteed that the insertion operations will be valid.



Ejercicios de Arrays o LinkedList



Example 1:

Input: nums = [0,1,2,3,4], index = [0,1,2,2,1]

Output: [0,4,1,3,2]

Explanation:

nums	index	target
0	0	[0]
1	1	[0,1]
2	2	[0,1,2]
3	2	[0,1,3,2]
4	1	[0,4,1,3,2]

Example 2:

Input: nums = [1,2,3,4,0], index = [0,1,2,3,0]

Output: [0,1,2,3,4]

Explanation:

nums	index	target
1	0	[1]
2	1	[1,2]
3	2	[1,2,3]
4	3	[1,2,3,4]
0	0	[0,1,2,3,4]

Example 3:

Input: nums = [1], index = [0]

Output: [1]

