

“Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de las heroicas batallas de Junín y Ayacucho”

**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**

**“La decana de América”**

**Facultad de Ingeniería de Sistemas e Informática**

**E.P. de Ingeniería de Software**



**Curso**

Análisis y Diseño de Algoritmos

**Alumno**

Gomez Olivas, Deyvi Pedro

**Profesora**

Chávez Soto Jorge Luis

**Lima, Perú**

**2024**

## FUNCIÓN QUE DIVIDE EL ARRAY EN DOS TOMADOS DE spacedBy ELEMENTOS

```
public static List<int []> divider (int[] arr, int spacedBy) {  
    int index = 0;  
    Vector arr1 = new Vector();  
    Vector arr2 = new Vector();  
    int size = arr.length;
```

```
    while (index < size) {  
        for (int i = 0; i < spacedBy; i++) {  
            if (index ≥ size) {  
                break;  
            }  
            arr1.add(arr[index]);  
            index++;  
        }  
        index += spacedBy;  
    }
```

For que se ejecuta  
spacedBy veces

Se suma a index  
los for realizados

Finalmente el while se ejecuta n  
veces donde n es el tamaño del arreglo

```
    index = spacedBy;  
    while (index < size) {  
        for (int i = 0; i < spacedBy; i++) {  
            if (index ≥ size) {  
                break;  
            }  
            arr2.add(arr[index]);  
            index++;  
        }  
        index += spacedBy;  
    }
```

```
    return List.of(arr1.getArray(), arr2.getArray());  
}
```

Entonces toda esta función es de complejidad  $O(n)$

## FUNCIÓN QUE UNE LOS DOS ARRAY ANTES SEPARADOS

```
public static int[] joiner (int[] first, int[] second, int each) {
    int totalSize = first.length + second.length;
    int iArr1 = 0, iArr2 = 0, index = 0;
    int[] joinedArray = new int [totalSize];
    int iter = roundDivision(Math.max(first.length, second.length), each);
    for (int i = 0; i < iter; i++) {
        iArr1 = i * each;
        iArr2 = i * each;
        Stack stack1 = new Stack();
        Stack stack2 = new Stack();
        Stack aux = new Stack();

        for (int j = iArr1; j < i*each + each; j++) {
            if (j ≥ first.length) {
                break;
            }
            aux.push(first[j]);
        }
        while (!aux.isEmpty()) {
            stack1.push(aux.pop().getDato());
        }
        aux = new Stack();
        for (int j = iArr2; j < i*each + each; j++) {
            if (j ≥ second.length) {
                break;
            }
            aux.push(second[j]);
        }
        while (!aux.isEmpty()) {
            stack2.push(aux.pop().getDato());
        }

        while (!stack1.isEmpty() || !stack2.isEmpty()) {
            if (stack1.peek() < stack2.peek()) {
                joinedArray[index] = stack1.pop().getDato();
                index++;
            } else {
                joinedArray[index] = stack2.pop().getDato();
                index++;
            }
        }
    }

    return joinedArray;
}
```

For que se repite  
n/each  
veces

For que se ejecuta each veces

While que se ejecuta each veces

For que se ejecuta each veces

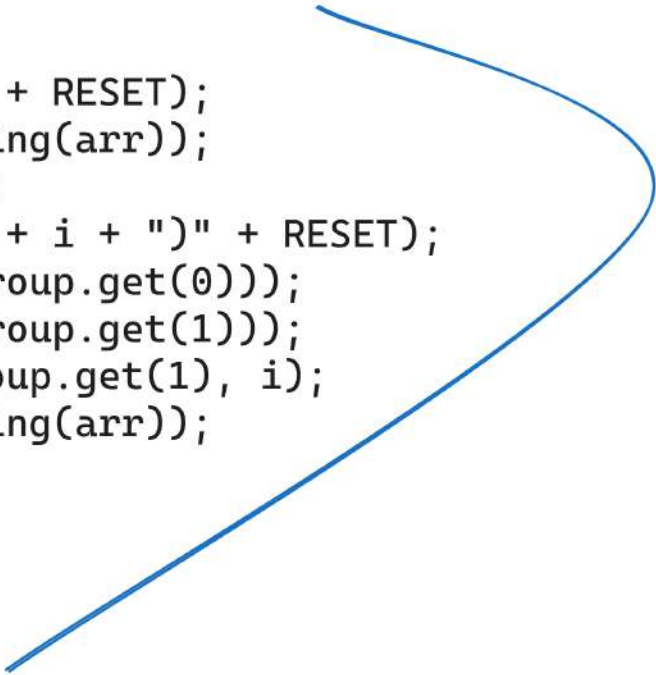
While que se ejecuta each veces

En el peor de los casos se repite este while  
hasta copiar todos los datos de las pilas a joinedArray  
Las pilas están con each elementos.  
Entonces el while en el peor de los casos se repite 2\*each veces

Por lo tanto la complejidad total de esta función es  $O(n)$

## FUNCION PRINCIPAL

```
public static void order (int[] arr) {  
    // Colores para la terminal  
    String RESET = "\u001B[0m";  
    String RED = "\u001B[31m";  
    String GREEN = "\u001B[32m";  
  
    int i = 1;  
    int index = 1;  
    System.out.println();  
    while (i < arr.length) {  
        System.out.println(RED + "Pasada: " + index + RESET);  
        System.out.println("Array: " + Arrays.toString(arr));  
n | List<int []> dividedGroup = divider(arr, i);  
        System.out.println(GREEN + "Particiones: (" + i + ")" + RESET);  
        System.out.println(Arrays.toString(dividedGroup.get(0)));  
        System.out.println(Arrays.toString(dividedGroup.get(1)));  
n | arr = joiner(dividedGroup.get(0), dividedGroup.get(1), i);  
        System.out.println("Fusion:" + Arrays.toString(arr));  
        index++;  
        i *= 2;  
    }  
    System.out.println();  
}
```

- 
- 1)  $i < n$
  - 2)  $i = 2^0, 2^1, 2^2, 2^3$
  - 3)  $\log n$  "veces"

Finalmente el algoritmo de ordenamiento tiene una complejidad de  $O(n \cdot \log n)$