Merge Sort

```java
public class MergeSortNumeros {

    // Método para dividir el arreglo y llamar a mergeSort recursivamente
    public static void mergeSort(int[] arr, int izquierda, int derecha) {
        if (izquierda < derecha) {
            // Encuentra el punto medio
            int medio = (izquierda + derecha) / 2;

            // Ordena la primera y la segunda mitad
            mergeSort(arr, izquierda, medio);
            mergeSort(arr, medio + 1, derecha);

            // Combina las mitades ordenadas
            merge(arr, izquierda, medio, derecha);
        }
```

División Recursiva del Arreglo (`mergeSort`):

- El método `mergeSort` es un método recursivo, lo que significa que se llama a sí mismo para realizar su tarea.
- Comienza con el arreglo completo y un rango definido por los índices `izquierda` y `derecha`.
- Calcula el punto medio `medio` del rango actual.
- Luego, divide el arreglo en dos mitades: desde `izquierda` hasta `medio`, y desde `medio + 1` hasta `derecha`.
- Llama a sí mismo para cada una de estas mitades, lo que significa que cada mitad se dividirá nuevamente hasta que cada segmento tenga solo un elemento.

```java
// Método para combinar dos subarreglos en un solo arreglo ordenado
public static void merge(int[] arr, int izquierda, int medio, int derecha) {
    // Tamaños de los subarreglos temporales
    int tam1 = medio - izquierda + 1;
    int tam2 = derecha - medio;

    // Arreglos temporales
    int[] Izq = new int[tam1];
    int[] Der = new int[tam2];

    // Copia los datos a los arreglos temporales
    for (int i = 0; i < tam1; ++i)
        Izq[i] = arr[izquierda + i];
    for (int j = 0; j < tam2; ++j)
        Der[j] = arr[medio + 1 + j];

    // Índices iniciales del primer y segundo subarreglo
    int i = 0, j = 0;

    // Índice inicial del subarreglo combinado
    int k = izquierda;
    while (i < tam1 && j < tam2) {
        if (Izq[i] <= Der[j]) {
            arr[k] = Izq[i];
            i++;
        } else {
            arr[k] = Der[j];
            j++;
        }
        k++;
```

1. **Combinación de Subarreglos (`merge`):**
   - Una vez que los subarreglos están divididos hasta el nivel de un solo elemento, comienza el proceso de combinación.
   - El método `merge` toma dos subarreglos ordenados (denominados `Izq` y `Der`) y los combina en un solo arreglo ordenado.
   - Crea dos arreglos temporales para almacenar los valores de los subarreglos y luego compara los elementos de estos arreglos de manera secuencial, colocando el menor elemento en el arreglo `arr`.

- Continúa este proceso hasta que todos los elementos de ambos subarreglos se hayan colocado en `arr`.
- Si uno de los subarreglos se vacía antes que el otro, simplemente copia los elementos restantes del otro subarreglo en `arr`.

```java
// Método principal para probar el algoritmo
Run | Debug
public static void main(String args[]) {
    int[] arreglo = { 9, 3, 1, 5, 13, 12 };

    System.out.println(x:"Arreglo original:");
    imprimirArreglo(arreglo);

    mergeSort(arreglo, izquierda:0, arreglo.length - 1);

    System.out.println(x:"\nArreglo ordenado:");
    imprimirArreglo(arreglo);
}

// Método para imprimir el arreglo
public static void imprimirArreglo(int[] arr) {
    for (int i = 0; i < arr.length; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
```

1. **El Método Principal** (`main`):
   - En el método `main`, se crea un arreglo de números desordenados.
   - Se llama al método `mergeSort` para ordenar el arreglo completo, pasando el índice inicial `0` y el índice final `arreglo.length - 1`.
   - Después de que el arreglo está ordenado, se imprime utilizando el método `imprimirArreglo`.

1. **Impresión del Arreglo** (`imprimirArreglo`):
   - Este es un método simple que recorre el arreglo y imprime cada elemento seguido de un espacio.