

# Ruleta 1: Merge Sort

## Descripción de trabajo

Para el desarrollo de este programa, se utilizó el lenguaje Java para crear un mecanismo de ordenamiento de datos utilizando el método merge sort.

Para comenzar, se realizó una breve investigación del método para entender mejor el funcionamiento y la implementación del merge, utilizándose como referencia el artículo de GeeksForGeeks<sup>[1]</sup>, el cual incluye un breve pseudocódigo que indica los pasos a seguir para una implementación exitosa.

Más adelante, se comenzó con el desarrollo del programa como tal, comenzando primero con el diseño del algoritmo y una vez terminado este, la implementación de una interfase gráfica. Se finalizó con una etapa de pruebas y debugging, así como con la realización de este reporte.

## Evidencia de funcionamiento

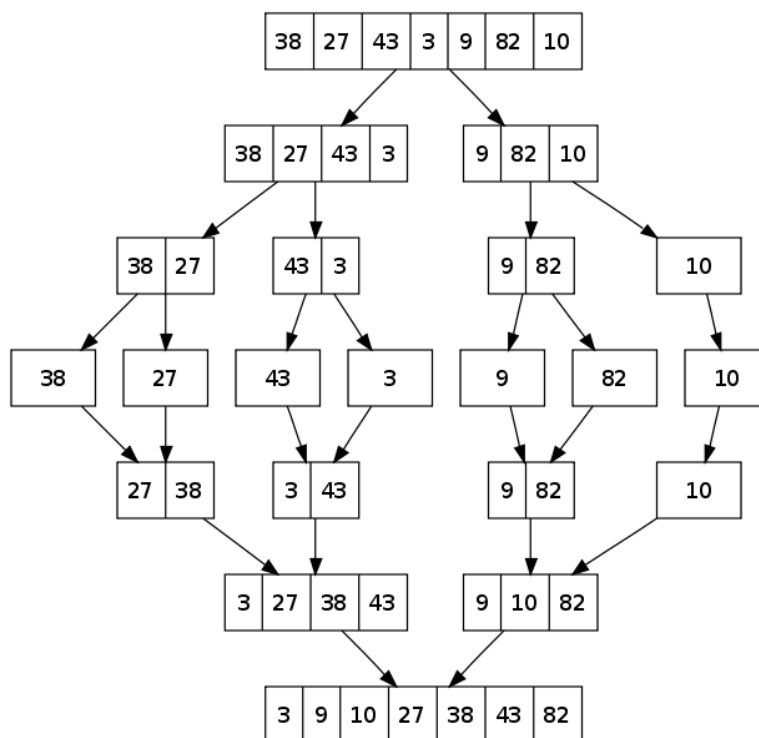
Durante la etapa de pruebas, se probó el funcionamiento del programa con diferentes entradas, incluyendo varios números enteros reales (positivos y negativos), números repetidos, y diversos tamaños de arreglos, desde 0 hasta 1M. En todas estas pruebas, el resultado fue exitoso, aunque el tiempo de ejecución varía lógicamente.

Sin embargo, es importante mencionar que este programa se diseñó para ordenar enteros, por lo que no es posible utilizarlo para ordenar doubles, longs, strings, ni ningún otro tipo de datos. También es necesario que el tamaño del arreglo sea por lo menos de 1 para garantizar una ejecución exitosa.

Si desea comprobar el funcionamiento del programa, puede obtener y compilar el código completo en GitHub<sup>[2]</sup>.

## Marco teórico

El algoritmo merge sort se basa en principio de "divide y vencerás". Este funciona por medio de la división de arreglos de datos hasta formar pequeños grupos que contengan de dos a tres elementos, los cuales pueden ser ordenados rápidamente, y una vez que estos pequeños grupos están ordenados, se juntan nuevamente en un arreglo general comparando los índices de cada mitad, de izquierda a derecha, ya que los valores más pequeños ahora se encontrarán al principio de cada arreglo. El siguiente diagrama ilustra el funcionamiento de un merge sort.



El tutorial de [GeeksForGeeks<sup>\[1\]</sup>](#) proponía que los pasos a seguir en la implementación eran los siguientes:

1. Encontrar el punto medio del arreglo de datos y partirlo en dos arreglos.
2. Llamar al método de ordenamiento para la primera mitad.
3. Llamar al método de ordenamiento para la segunda mitad.
4. Juntar ambas mitades en orden.

El programa se creó a partir de estos pasos, utilizando recursividad del método de ordenamiento, hasta que se llegara a tener un arreglo de tamaño dos o tres, los cuales son ordenados con bubble sort, y una vez teniendo estos elementos ordenados, se puede proceder a ordenar y juntar a un arreglo general.

## Código

```
1. public int[] mergeSort(int[] numbers){
2.
3.     if(numbers.length > 3){
4.
5.         int arrayMiddleA, arrayMiddleB;
6.
7.         if(numbers.length % 2 == 0){
8.             arrayMiddleA = numbers.length / 2;
9.             arrayMiddleB = arrayMiddleA;
10.        }
11.        else {
12.            arrayMiddleA = (numbers.length + 1) / 2;
13.            arrayMiddleB = arrayMiddleA - 1;
14.        }
15.
16.        int tmpFirstHalf[] = new int[arrayMiddleA];
17.        int tmpSecondHalf[] = new int[arrayMiddleB];
18.        System.arraycopy(numbers, 0, tmpFirstHalf, 0, arrayMiddleA);
19.        System.arraycopy(numbers, arrayMiddleA, tmpSecondHalf, 0, arrayMiddleB);
20.
21.        tmpFirstHalf = mergeSort(tmpFirstHalf).clone();
22.        tmpSecondHalf = mergeSort(tmpSecondHalf).clone();
23.
24.        int a = 0, b = 0, c = 0;
25.
26.        merge: for(int i=0; i<numbers.length; i++){
27.            if (tmpFirstHalf[a] < tmpSecondHalf[b]) {
28.                numbers[i] = tmpFirstHalf[a];
29.                a++;
30.            }
31.            else {
32.                numbers[i] = tmpSecondHalf[b];
33.                b++;
34.            }
35.
36.            if (a == tmpFirstHalf.length || b == tmpSecondHalf.length) {
37.                c = i+1;
38.                break merge;
            }
```

```

39.     }
40. }
41.
42.     for (int i=c; i<numbers.length; i++) {
43.         if (a==tmpFirstHalf.length) {
44.             numbers[i] = tmpSecondHalf[b];
45.             b++;
46.         }
47.         else if (b==tmpSecondHalf.length) {
48.             numbers[i] = tmpFirstHalf[a];
49.             a++;
50.         }
51.     }
52.     return numbers;
53. }
54.
55.     else {
56.         for (int i=0; i<numbers.length; i++) {
57.             for(int j=1; j<numbers.length-i; j++) {
58.                 if (numbers[j-1] > numbers[j]) {
59.                     int tmp = numbers[j];
60.                     numbers[j] = numbers[j-1];
61.                     numbers[j-1] = tmp;
62.                 }
63.             }
64.         }
65.         return numbers;
66.     }
67.
68. }

```

### Comentarios del código por líneas:

1-68: Método mergeSort, que recibe un arreglo de enteros como parámetro.

3-53: Si el tamaño del arreglo es mayor que 3, se ejecuta esta primera condición.

7-14: Determina la mitad del arreglo, y en caso de que el tamaño del arreglo no sea par, la primera mitad será de tamaño  $(n+1)/2$ .

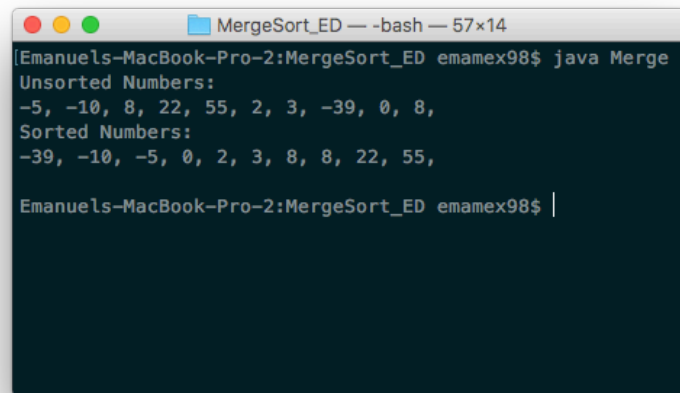
16-19: Declaración de los arreglos temporales que contendrán cada mitad de los datos.

21-22: Se llama al mismo método (recursividad) para continuar dividiendo/ordenando los arreglos respectivamente.

26-40: Se unen las dos mitades al arreglo general, comparando elementos de izquierda a derecha para que se unan en orden.

52: Regresa un arreglo de tamaño menor o igual a 3 ordenado.

parte posterior. También se llevaron a cabo otras pruebas donde se ingresan números negativos, que se pueden observar en la captura de pantalla de la terminal a continuación:



```
Emanuels-MacBook-Pro-2:MergeSort_ED emamex98$ java Merge
Unsorted Numbers:
-5, -10, 8, 22, 55, 2, 3, -39, 0, 8,
Sorted Numbers:
-39, -10, -5, 0, 2, 3, 8, 8, 22, 55,

Emanuels-MacBook-Pro-2:MergeSort_ED emamex98$
```

Si desea realizar sus propias pruebas utilizando el código que se encuentra disponible en GitHub<sup>[2]</sup> se proponen los siguientes casos:

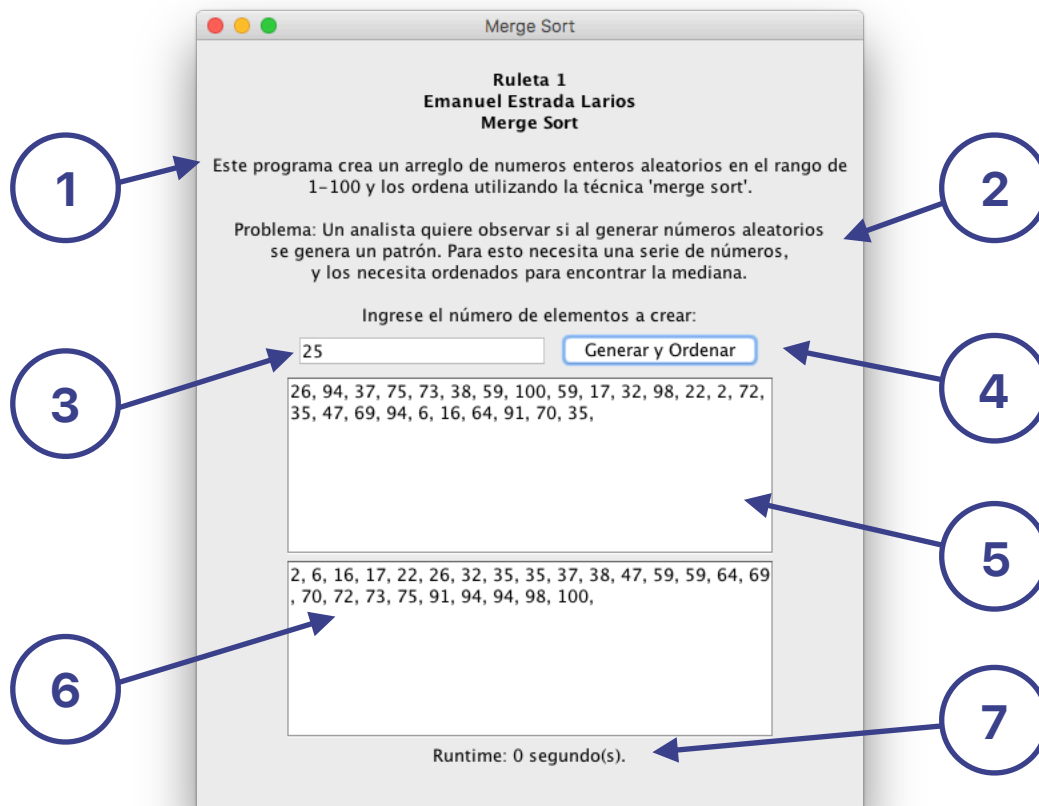
1. Probar el programa utilizando la interface gráfica asignando diferentes valores al tamaño del arreglo de números a crear, ingresando un número entre 1 y 1,000,000, evitando ingresar valores menores a 1 o que no sean de tipo entero. El tiempo de ejecución puede variar dependiendo de la capacidad de cada computadora.
2. Utilizar el main que se encuentra comentado en la parte posterior de la clase Merge, en donde se pueden asignar valores personalizados al arreglo "numbers", evitando incluir valores no-enteros.
3. Durante las pruebas, no se excedió el límite de 1,000,000 de elementos en el arreglo. Se recomienda no exceder este límite para una evitar saturar la memoria.

## Argumentos

Este programa resuelve el problema propuesto de manera óptima ya que el tiempo de ejecución es mucho menor al que tomarían otros métodos. Por ejemplo, si se compara el 'merge sort' con un simple 'bubble sort' o un 'insertion sort', este es 50 veces más rápido en promedio, ya que el merge sort tiene un runtime promedio de  $O(n \log n)$ , mientras que los otros dos métodos tienen uno de  $O(n^2)$ <sup>[3]</sup>.

Además, debido a que se utiliza un método de eliminación y no de comparación de cada índice con todos los demás, se optimiza el uso de memoria, y es un método estable.

## Descripción de la GUI



1. Descripción del programa.
2. Presentación del problema y aplicación de la vida real.
3. Campo de texto donde el usuario ingresa el número de datos que desea crear.
4. Botón que crea el arreglo de tamaño especificado por el usuario y lo ordena.
5. Area de texto donde se imprimen los elementos desordenados.
6. Area de texto donde se imprimen los elementos ordenados.
7. Temporizador donde se muestra el tiempo de ejecución en segundos.

## Referencias y Recursos:

- [1] "Merge Sort" by Geeks for Geeks: <http://www.geeksforgeeks.org/merge-sort/>
- [2] GitHub repository by Emanuel Estrada: [https://github.com/emamex98/MergeSort\\_ED](https://github.com/emamex98/MergeSort_ED)
- [3] "Sorting Algorithms" by K. Moore & G. Pilling: <https://brilliant.org/wiki/sorting-algorithms/#sorting-algorithms>