MERGE SORT

Merge Sort este un algoritm de sortare eficient și stabil care folosește o strategie de divizare și stăpânire (divide and conquer) pentru a sorta un vector sau o listă de elemente. A fost dezvoltat de către John von Neumann în 1945 și a fost îmbunătățit ulterior de către alți cercetători.

Iată pașii principali ai algoritmului Merge Sort:

Divizare: Vectorul nesortat este divizat în două jumătăți egale.

Conducere: Fiecare jumătate este sortată recursiv prin aplicarea aceleiași proceduri de divizare și stăpânire.

Combinație: Cele două jumătăți sortate sunt unite împreună într-un vector sortat. Această operație de combinare asigură menținerea ordinii corecte a elementelor.

Această abordare duce la obținerea unei complexități a timpului de execuție de O(n log n), unde "n" este numărul de elemente din vector. Deși Merge Sort nu are cel mai mic factor constant și utilizează spațiu suplimentar pentru stocarea temporară a elementelor, este apreciat pentru stabilitatea sa, eficiența în cazul listelor înlănțuite și capacitatea sa de a funcționa eficient în cazul unor seturi de date mari.

Algoritmul Merge Sort poate fi utilizat și pentru a sorta structuri de date complexe, cum ar fi liste dublu-înlănțuite sau matrice bidimensionale, dar implementarea sa clasică este de obicei aplicată pentru vectori.

```java
import java.util.Scanner;

public class MergeSortWithInput {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Introdu dimensiunea vectorului: ");
            int size = scanner.nextInt();

            int[] array = new int[size];

            System.out.println("Introdu elementele vectorului:");

            for (int i = 0; i < size; i++) {
                System.out.print("Elementul " + (i + 1) + ": ");
                array[i] = scanner.nextInt();
```

```
        }

        System.out.println("Vectorul inainte de sortare:");
        printArray(array);

        mergeSort(array, 0, array.length - 1);

        System.out.println("Vectorul dupa sortare:");
        printArray(array);
    } finally {
        // Închide Scanner-ul în blocul finally pentru a te asigura că se închide chiar dacă apare o excepție.
        scanner.close();
    }
}

static void mergeSort(int array[], int left, int right) {
    if (left < right) {
        // Găsește mijlocul vectorului
        int middle = (left + right) / 2;

        // Sortează jumătatea stângă și dreaptă
        mergeSort(array, left, middle);
        mergeSort(array, middle + 1, right);

        // Unește părțile sortate
        merge(array, left, middle, right);
    }
}

static void merge(int array[], int left, int middle, int right) {
    int n1 = middle - left + 1;
    int n2 = right - middle;

    // Crează vectoruri temporare
    int leftArray[] = new int[n1];
    int rightArray[] = new int[n2];

    // Copiază datele în vectorurile temporare
    for (int i = 0; i < n1; ++i)
        leftArray[i] = array[left + i];
    for (int j = 0; j < n2; ++j)
        rightArray[j] = array[middle + 1 + j];

    // Unește vectorii temporari

    // Indicii inițiali ai vectorilor temporari
    int i = 0, j = 0;

    // Indexul inițial al vectorului unit
    int k = left;
```

```
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                array[k] = leftArray[i];
                i++;
            } else {
                array[k] = rightArray[j];
                j++;
            }
            k++;
        }

        // Copiază elementele rămase din leftArray[] (dacă există)
        while (i < n1) {
            array[k] = leftArray[i];
            i++;
            k++;
        }

        // Copiază elementele rămase din rightArray[] (dacă există)
        while (j < n2) {
            array[k] = rightArray[j];
            j++;
            k++;
        }
    }

    static void printArray(int array[]) {
        int n = array.length;
        for (int i = 0; i < n; ++i)
            System.out.print(array[i] + " ");
        System.out.println();
    }
}
```

Afisare

```
Introdu dimensiunea vectorului: 4
Introdu elementele vectorului:
Elementul 1: 7
Elementul 2: 4
Elementul 3: 6
Elementul 4: 2
Vectorul inainte de sortare:
7 4 6 2
Vectorul dupa sortare:
2 4 6 7
```