Introducción

En este proyecto vemos como el programa es capaz de leer archivos tipo CSV y que muestre las gráficas de los datos. Además este utiliza 6 diferentes tipos de ordenamiento que lo ordena de Ascendente o Descendente y además de la generación de un reporte de PDF con los datos desordenados y ordenados. Esta práctica integra el modelo MVC (Modelo, Vista y Controlador) para que se más ordenado la estructura del Código

Requerimientos del Sistema

Sistema Operativo: Windows, Linux o macOS.

Java Runtime Environment (JRE): Versión 8 o superior.

Memoria RAM: Mínimo 2 GB recomendados.

Espacio en disco: 100 MB libres.

Estructura del Proyecto

El proyecto sigue el patrón MVC (Modelo-Vista-Controlador) y está organizado en los siguientes paquetes:

Paquete Model

Clase Datos:

Gestiona la carga y manipulación de datos desde archivos .ipcd1.

Implementa los algoritmos de ordenamiento (Burbuja, Inserción, Selección, QuickSort, MergeSort, ShellSort).

Mantiene los datos originales y ordenados.

Proporciona estadísticas (pasos, comparaciones, intercambios, tiempo de ejecución).

Paquete View

Clase View:

Interfaz gráfica construida con Swing.

Muestra la gráfica de datos usando JFreeChart.

Proporciona controles para seleccionar algoritmo, dirección y velocidad de ordenamiento.

Muestra estadísticas en tiempo real durante el ordenamiento.

Paquete Controller

Clase Controlador:

Gestiona la interacción entre la vista y el modelo.

Implementa listeners para los botones: Buscar, Ordenar y Generar PDF.

Usa hilos (Thread) para el ordenamiento asíncrono.

Paquete practica2

Clase PDFGenerator:

Genera reportes en PDF con:

Portada, información del proceso, datos originales, gráficas (ordenada y desordenada).

Usa la librería iTextPDF para crear el documento.

Métodos Clave

Bubble Sort

Propósito: Ordenar un array mediante comparaciones e intercambios adyacentes.

Algoritmo: Recorre el array desde el primer elemento hasta el penúltimo. Compara cada elemento con el siguiente:

Si están en orden incorrecto (según dirección ascendente/descendente), los intercambia.

Repite el proceso hasta que no se requieran más intercambios.

```
    public void ordenarBurbuja(boolean ascendente) {
        pasos = 0;
        comparaciones = 0;
        intercambios = 0;
        tiempoInicio = System.currentTimeMillis();
        for (int i = 0; i < conteo.length - 1; i++) {
            for (int j = 0; j < conteo.length - i - 1; j++) {
                if ((ascendente && conteo[j] > conteo[j + 1]) || (!ascendente && conteo[j] < conteo[j + 1])) {
                    intercambiar(j, j + 1);
                    notificarPaso(j, j + 1, "Intercambio");
                } else {
                    notificarPaso(j, j + 1, "Comparación");
                }
            }
        }
}
```

Insert Sort

Propósito: Construir una secuencia ordenada un elemento a la vez.

Algoritmo:

Divide el array en una parte ordenada y otra sin ordenar.

Toma el primer elemento de la parte sin ordenar y lo inserta en la posición correcta dentro de la parte ordenada.

Repite hasta que todo el array esté ordenado.

```
    public void ordenarInsercion(boolean ascendente) {
        pasos = 0;
        comparaciones = 0;
        intercambios = 0;
        tiempoInicio = System.currentTimeMillis();
        for (int i = 1; i < conteo.length; i++) {
            int key = conteo[i];
            String keyCat = categoria[i];
            int j = i - 1;
            while (j >= 0 && ((ascendente && conteo[j] > key) || (!ascendente && conteo[j] < key))) {
                comparaciones++;
                conteo[j + 1] = conteo[j];
                categoria[j + 1] = categoria[j];
                j--;
                pasos++;
                notificarPaso(j, i, "Desplazamiento");
            }
            conteo[j + 1] = key;
            categoria[j + 1] = keyCat;
        }
    }
```

Select Sort

Propósito: Seleccionar repetidamente el elemento mínimo/máximo y colocarlo en su posición correcta.

Algoritmo:

Busca el elemento mínimo (o máximo) en el array sin ordenar.

Intercámbialo con el primer elemento sin ordenar.

Repite el proceso para el resto del array.

```
public void ordenarSeleccion(boolean ascendente) {
    pasos = 0;
    comparaciones = 0;
    intercambios = 0;
    tiempoInicio = System.currentTimeMillis();
    for (int i = 0; i < conteo.length - 1; i++) {
        int extremo = i;
        for (int j = i + 1; j < conteo.length; j++) {
            if ((ascendente && conteo[j] < conteo[extremo]) || (!ascendente && conteo[j] > conteo[extremo])) {
                extremo = j;
            }
            pasos++;
            notificarPaso(j, extremo, "Comparación");
        }
        intercambiar(i, extremo);
    }
}
```

Quick Sort

Propósito: Ordenar mediante división y conquista, usando un pivote.

Algoritmo:

Partición: Selecciona un pivote y reorganiza el array para que:

Elementos < pivote estén a su izquierda.

Elementos > pivote estén a su derecha.

Recursión: Aplica QuickSort a las subarrays izquierda y derecha.

```java
public void ordenarQuickSort(boolean ascendente) {
    pasos = 0;
    tiempoInicio = System.currentTimeMillis();
    quickSort(0, conteo.length - 1, ascendente);
}

private void quickSort(int inicio, int fin, boolean ascendente) {
    if (inicio < fin) {
        int indiceParticion = particion(inicio, fin, ascendente);
        quickSort(inicio, indiceParticion - 1, ascendente);
        quickSort(indiceParticion + 1, fin, ascendente);
    }
}

private int particion(int inicio, int fin, boolean ascendente) {
    int pivote = conteo[fin];
    int i = inicio - 1;
    for (int j = inicio; j < fin; j++) {
        if ((ascendente && conteo[j] <= pivote) || (!ascendente && conteo[j] >= pivote)) {
            i++;
            intercambiar(i, j);
            notificarPaso(i, j, "Partición");
        }
    }
    intercambiar(i + 1, fin);
    return i + 1;
}
```

Merge Sort

Propósito: Ordenar dividiendo el array en mitades, ordenándolas y fusionándolas.

Algoritmo:

División: Divide el array en dos mitades.

Conquista: Ordena cada mitad recursivamente.

Combinación: Fusiona las mitades ordenadas.

```java
public void ordenarMergeSort(boolean ascendente) {
    pasos = 0;
    tiempoInicio = System.currentTimeMillis();
    mergeSort(0, conteo.length - 1, ascendente);
}

private void mergeSort(int inicio, int fin, boolean ascendente) {
    if (inicio < fin) {
        int medio = (inicio + fin) / 2;
        mergeSort(inicio, medio, ascendente);
        mergeSort(medio + 1, fin, ascendente);
        merge(inicio, medio, fin, ascendente);
    }
}

private void merge(int inicio, int medio, int fin, boolean ascendente) {
    int n1 = medio - inicio + 1;
    int n2 = fin - medio;

    int[] L = new int[n1];
    int[] R = new int[n2];
    String[] Lcat = new String[n1];
    String[] Rcat = new String[n2];

    System.arraycopy(conteo, inicio, L, 0, n1);
    System.arraycopy(conteo, medio + 1, R, 0, n2);
    System.arraycopy(categoria, inicio, Lcat, 0, n1);
    System.arraycopy(categoria, medio + 1, Rcat, 0, n2);

    int i = 0, j = 0, k = inicio;
    while (i < n1 && j < n2) {
        if ((ascendente && L[i] <= R[j]) || (!ascendente && L[i] >= R[j])) {
            conteo[k] = L[i];
            categoria[k] = Lcat[i];
            i++;
        } else {
            conteo[k] = R[j];
```

Shell Sort

Propósito: Mejora del Insertion Sort comparando elementos distantes.

Algoritmo:

Define una secuencia de brechas (gaps).

Para cada brecha, aplica Insertion Sort a subarrays espaciados.

Reduce la brecha y repite hasta que la brecha sea 1.

```java
public void ordenarShellSort(boolean ascendente) {
    pasos = 0;
    tiempoInicio = System.currentTimeMillis();
    int n = conteo.length;
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = conteo[i];
            String tempCat = categoria[i];
            int j;
            for (j = i; j >= gap && ((ascendente && conteo[j - gap] > temp) || (!ascendente && conteo[j - gap] < temp)); j -
                conteo[j] = conteo[j - gap];
                categoria[j] = categoria[j - gap];
                pasos++;
                notificarPaso(j, j - gap, "Shell Sort");
            }
            conteo[j] = temp;
            categoria[j] = tempCat;
        }
    }
}
```

Dependencias

Librerías externas:

itextpdf (v5.5.10): Para generación de PDF.

jfreechart (v1.5.0): Para gráficas.

Conclusión

En este proyecto se aplicaron hilos para la creación de los ordenamientos además de usar Threads.sleep para darle tiempo al algoritmo para que se ejecute. Además de la creación del PDF y se ha utilizado todos los conocimiento en el Laboratorio. Además que emplea de manera directa otras librerías para la creación de gráficos a partir de datos ya implementados en un tipo de archivo especial y para la creación de PDF y la implementación modelo MVC