Reporte Técnico de Actividades Práctico-Experimentales Nro. 001

1. Datos de Identificación del Estudiante y la Práctica

Nombre del estudiante(s)	Francis Valdiviezo Derick Vargas
Asignatura	Estructura de Datos
Ciclo	Tercero A
Unidad	2
Resultado de aprendizaje de la unidad	Aplica los métodos de ordenación y búsqueda en la resolución de problemas, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Práctica Nro.	001
Título de la Práctica	Ordenación básica en Java: Burbuja, Selección e Inserción
Nombre del Docente	Ing. Andres Navas
Fecha	Jueves 20 de noviembre Viernes 21 de noviembre
Horario	07h30 – 10h30 07h30 – 09h30
Lugar	Aula 334
Tiempo planificado en el Sílabo	5 horas

2. Objetivo(s) de la Práctica

Ejecutar y analizar comparativamente los algoritmos de Burbuja, Selección e Inserción sobre casos de prueba, para determinar cuándo conviene cada uno en función de tamaño, grado de orden y duplicados.

3. Materiales, Reactivos, Equipos y Herramientas

- Guía de pruebas con datasets y salidas esperadas.
- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión “Extension Pack for Java”) o IntelliJ IDEA Community.
- Sistema de control de versiones: Git; repositorio en GitHub. • EVA/Moodle institucional: para entrega de evidencias.
- Herramientas de documentación: README Markdown, editor ofimático (Google Docs/LibreOffice/Word).

4. Procedimiento / Metodología Ejecutada

- Para la realización de esta práctica comenzamos con el análisis de el documento presentado para el desarrollo de la practica
- A continuación, se desarrollo una breve investigación sobre las comparaciones de los métodos de ordenamiento.
- Con los resultados del taller anterior, comenzamos a modificar las partes necesarias para que tenga un correcto funcionamiento.
- Mediante el uso correcto de la ia, nos guiamos para realizar una clase en la cual podamos generar todos los datasets necesarios.
- Realizamos la implementación de lectura de los dataset dentro de nuestro proyecto y de todos los cambios necesarios como el contador de comparisons y los swaps.
- Analizamos que los resultados de los algoritmos sea los indicados y cargamos en el repositorio de github
- Y como ultimo paso se dio paso a la realización de este informe.

5. Resultados

Link del repositorio (Trabajo manejado en la rama tallerComparaciones): https://github.com/FrancisValdiviezoUNL/Ordenamiento-Estructura/tree/tallerComparaciones

- **Clase para generar los dataset**

```java
package ed.u2.sorting.datasets;

import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.*;

public class DatasetGenerator {

    private static final Random rnd = new Random(42);

    private static final String[] APELLIDOS_30 = {
        "Guerrero","Naranjo","Cedeño","Márquez","Valdiviezo","Ramírez","Lozano","Vera",
        "Carvajal","López","Paredes","Ríos","Sánchez","Mora","Aguirre","Reyes","Romero",
        "Vega","González","Jiménez","Palacios","Roldán","Mendoza","Ortega","Campos",
        "Cárdenas","Fajardo","Jara","Solano","Castro"
    };

    private static final String[] INSUMOS = {
        "Guante Nitrilo Talla M", "Alcohol 70% 1L", "Gasas 10x10", "Jeringa 5ml", "Venda Elástica 10cm",
        "Mascarilla KN95", "Termómetro Digital", "Algodón 250g", "Tijeras Médicas", "Analgésico Genérico"
    };

    public static void generarCitas100(String outputPath) throws IOException {
        try (FileWriter fw = new FileWriter(outputPath)) {
            fw.write("id;apellido;fechaHora\n");

            LocalDate start = LocalDate.of(2025, 3, 1);
            LocalDate end = LocalDate.of(2025, 3, 31);
            LocalTime open = LocalTime.of(8, 0);
            LocalTime close = LocalTime.of(18, 0);

            for (int i = 1; i <= 100; i++) {
                String id = String.format("CITA-%03d", i);

                String apellido = APELLIDOS_30[rnd.nextInt(APELLIDOS_30.length)];

                long days = end.toEpochDay() - start.toEpochDay();
                long randomDay = start.toEpochDay() + rnd.nextInt((int) days + 1);
                LocalDate d = LocalDate.ofEpochDay(randomDay);

                int minutesRange = (int) (close.toSecondOfDay() - open.toSecondOfDay()) / 60;
                int randomMinutes = rnd.nextInt(minutesRange + 1);
                LocalTime t = open.plusMinutes(randomMinutes);

                LocalDateTime fechaHora = d.atTime(t.getHour(), t.getMinute());

                fw.write(id + ";" + apellido + ";" + fechaHora.toString() + "\n");
            }
        }
        System.out.println("citas.csv generado en " + outputPath);
    }

    public static void generarCitas100CasiOrdenadas(String inputPath, String outputPath) throws IOException {
        List<String> lines = java.nio.file.Files.readAllLines(java.nio.file.Paths.get(inputPath));
        if (lines.isEmpty()) return;

        List<String> data = new ArrayList<>(lines.subList(1, lines.size()));

        data.sort(Comparator.comparing(o -> o.split(";")[2]));

        // exactamente 5 swaps
        int n = data.size();
        Set<String> usedPairs = new HashSet<>();
        int swaps = 0;

        while (swaps < 5) {
            int a = rnd.nextInt(n);
            int b = rnd.nextInt(n);
            if (a == b) continue;
            String key = a < b ? a + "-" + b : b + "-" + a;
            if (usedPairs.contains(key)) continue;
            Collections.swap(data, a, b);
            usedPairs.add(key);
            swaps++;
        }

        try (FileWriter fw = new FileWriter(outputPath)) {
            fw.write("id;apellido;fechaHora\n");
            for (String s : data) {
                fw.write(s + "\n");
            }
        }
        System.out.println("citas_casiOrdenadas.csv generado en " + outputPath);
    }

    public static void generarPacientes500(String outputPath) throws IOException {
        try (FileWriter fw = new FileWriter(outputPath)) {
            fw.write("id;apellido;prioridad\n");

            List<String> pool = new ArrayList<>();

            List<String> grupoA = Arrays.asList("Ramírez", "Guerrero", "Mendoza", "Cedeño", "Carrillo");
            List<String> grupoB = Arrays.asList("Aguirre", "Márquez", "López", "Ríos", "Valdiviezo");
            List<String> grupoC = Arrays.asList("Vega", "González", "Reyes", "Ortega", "Romero");
    }
```

- **En la estructura del proyecto tenemos 4 package**
  - **Dataset**
    - DatasetGenerator (Clase para generar los dataset)(Clase independiente)
    - DatasetLoader (Clase para cargar los dataset al programa)
  - **Modelos**
    - Cita (Clase para manejar los datos del dataset cita)
    - Paciente (Clase para manejar los datos del dataset paciente)
    - ProductoInventario (Clase para manejar el dataset de inventario)
  - **Ordenamiento**
    - BubbleSort (Método de ordenamiento burbuja)
    - InsetionSort (Método de ordenamiento de Inserción)
    - SelectionSort (Método de ordenamiento de Selección)
    - SortingMetrics (Método fundamental para guardar los tiempos de ejecución de cada método)
  - **Util**
    - CSVloader (Clase fundamental para cargar los archios csv al programa)
  - Clase main (Clase ejecutora del programa)

- **Resultados Obtenidos**

```
=== Taller Ordenación - Comparación Básica ===
1. Probar citas por fecha
2. Probar citas casi ordenadas por fecha
3. Probar pacientes por apellido
4. Probar inventario por stock
5. Salir
Opción: 1
Dataset: citas.csv (n=100) - clave = fechaHora
------------------------------
Burbuja:
    Tiempo(ms)=2.114
    Comparaciones=4947
    Swaps=2529
------------------------------
Seleccion:
    Tiempo(ms)=1.1775
    Comparaciones=4950
    Swaps=93
------------------------------
Insercion:
    Tiempo(ms)=0.6696
    Comparaciones=2623
    Swaps=2529
------------------------------
El algoritmo con menos swaps fue: Selección con 93 swaps.

=== Taller Ordenación - Comparación Básica ===
1. Probar citas por fecha
2. Probar citas casi ordenadas por fecha
3. Probar pacientes por apellido
4. Probar inventario por stock
5. Salir
Opción: 2
Dataset: citas_casiOrdenadas.csv (n=100) - clave = fechaHora
------------------------------
Burbuja:
    Tiempo(ms)=0.5822
    Comparaciones=3354
    Swaps=197
------------------------------
Seleccion:
    Tiempo(ms)=0.8814
    Comparaciones=4950
    Swaps=5
------------------------------
Insercion:
    Tiempo(ms)=0.0637
    Comparaciones=296
    Swaps=197
------------------------------
El algoritmo con menos swaps fue: Selección con 5 swaps.

=== Taller Ordenación - Comparación Básica ===
1. Probar citas por fecha
2. Probar citas casi ordenadas por fecha
3. Probar pacientes por apellido
4. Probar inventario por stock
5. Salir
Opción: 3
Dataset: pacientes.csv (n=500) - clave = apellido
------------------------------
Burbuja:
    Tiempo(ms)=17.9758
    Comparaciones=124315
    Swaps=59779
------------------------------
Seleccion:
    Tiempo(ms)=11.9287
    Comparaciones=124750
    Swaps=484
------------------------------
Insercion:
    Tiempo(ms)=11.4879
    Comparaciones=60278
    Swaps=59779
------------------------------
El algoritmo con menos swaps fue: Selección con 484 swaps.

=== Taller Ordenación - Comparación Básica ===
1. Probar citas por fecha
2. Probar citas casi ordenadas por fecha
3. Probar pacientes por apellido
4. Probar inventario por stock
5. Salir
Opción: 4
Dataset: inventario_inverso.csv (n=500) - clave = stock
------------------------------
Burbuja:
    Tiempo(ms)=39.814
    Comparaciones=124750
    Swaps=124750
------------------------------
Seleccion:
    Tiempo(ms)=20.3772
    Comparaciones=124750
    Swaps=250
------------------------------
Insercion:
    Tiempo(ms)=2.4732
    Comparaciones=124750
    Swaps=124750
------------------------------
El algoritmo con menos swaps fue: Selección con 250 swaps.
```

6. Preguntas de Control

- **¿Por qué imprimir trazas durante la medición distorsiona los tiempos?**

El imprimir en la consola, en si es el proceso computacionalmente más costoso y sobre todo muy lento, más que las operaciones en memoria, que realizan los algoritmos de ordenación.

Si se mantienen las impresiones dentro del bucle de medición, el tiempo registrado incluirá la latencia de escribir en pantalla, lo cual "distorsiona" la medición real del rendimiento del algoritmo.

- **Explica por qué Selección tiene comparaciones ~ n(n−1)/2 sin importar el orden inicial.**

El algoritmo de selección, se rige y funciona buscando el elemento más pequeño, o mayor, de la lista para colocarlo en la ubicación correcta.
Para encontrar ese elemento, el algoritmo siempre tiene q recorrer todo lo que no está ordenado del arreglo, sin importar si ya están ordenados o no

- **¿Por qué Inserción es competitivo en datos casi ordenados?**

El método de inserción es demasiado eficiente por el simple hecho de que, al tomar un elemento, solo necesita compararlo y de ahí intercambiarlo con los anteriores encontrando su lugar.

- **¿Qué papel juegan los duplicados en la estabilidad del resultado?**

Los duplicados ponen a prueba la estabilidad del algoritmo, que es la capacidad de mantener el orden relativo original de registros que tienen la misma clave.

- **¿Por qué Burbuja con corte temprano mejora en “casi ordenado” pero no en “inverso”?**

En "Casi Ordenado": El "corte temprano" es el q detiene el algoritmo por si pasa por todo el arreglo sin hacer ningún intercambio. Si está casi ordenado, esto ocurre muy rápido, reduciendo drásticamente el tiempo.

En "Inverso": El elemento más grande está al principio y debe moverse hasta el final, y el más pequeño está al final y debe avanzar una posición por cada pasada completa. Esto obliga a realizar intercambios en cada iteración hasta el final, impidiendo que el corte temprano se active, por lo que Burbuja "penaliza" este escenario.

7. Conclusiones

En conclusión, mediante la realización de este taller logramos identificar como realizar comparaciones entre métodos de ordenamiento, al igual de cómo utilizar las clases para cronometrar el tiempo de ejecución.

La eficiencia de la ordenación depende directamente del orden inicial de los datos. El algoritmo de Inserción es adaptable y superior en listas casi ordenadas, mientras que Selección tiene un comportamiento rígido y constante en comparaciones. Es crucial aislar las operaciones de Entrada/Salida (impresiones) durante la ejecución para obtener mediciones de tiempo válidas

8. Recomendaciones

Como recomendación tenemos que tenemos que probar con diferentes casos, ya que cada algoritmo de ordenamiento puede interactuar de diferente manera con distintos casos.

Se recomienda utilizar el algoritmo de Inserción para listas pequeñas o parcialmente ordenadas, ya que minimiza los movimientos y mantiene la estabilidad de los datos (respeta el orden de duplicados). Por otro lado, se sugiere usar Selección únicamente cuando sea crítico minimizar el número de intercambios (escrituras en memoria).