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
Unidad	2
Resultado de aprendizaje de la unidad	
Práctica Nro.	001
Título de la Práctica	Taller 5 Ordenamiento en java
Nombre del Docente	Ing. Andres Navas
Fecha	17/11/2025
Horario	
Lugar	
Tiempo planificado en el Sílabo	

2. Objetivo(s) de la Práctica

Implementar y comparar tres algoritmos de ordenación in-place sobre arreglos pequeños, y validar su funcionamiento con trazas y casos de prueba reproducibles.

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

Como primer paso se realizo una investigación sobre los métodos de ordenamiento.

Seguidamente comenzamos a realizar la implementación de códigos de cada método de ordenamineto.

Se genero una clase de main para ejecutar todos los métodos.

Después de probar que todo funcione y este bien, se creo un repositorio en github y se sube todo el proyecto incluido el readme.md.

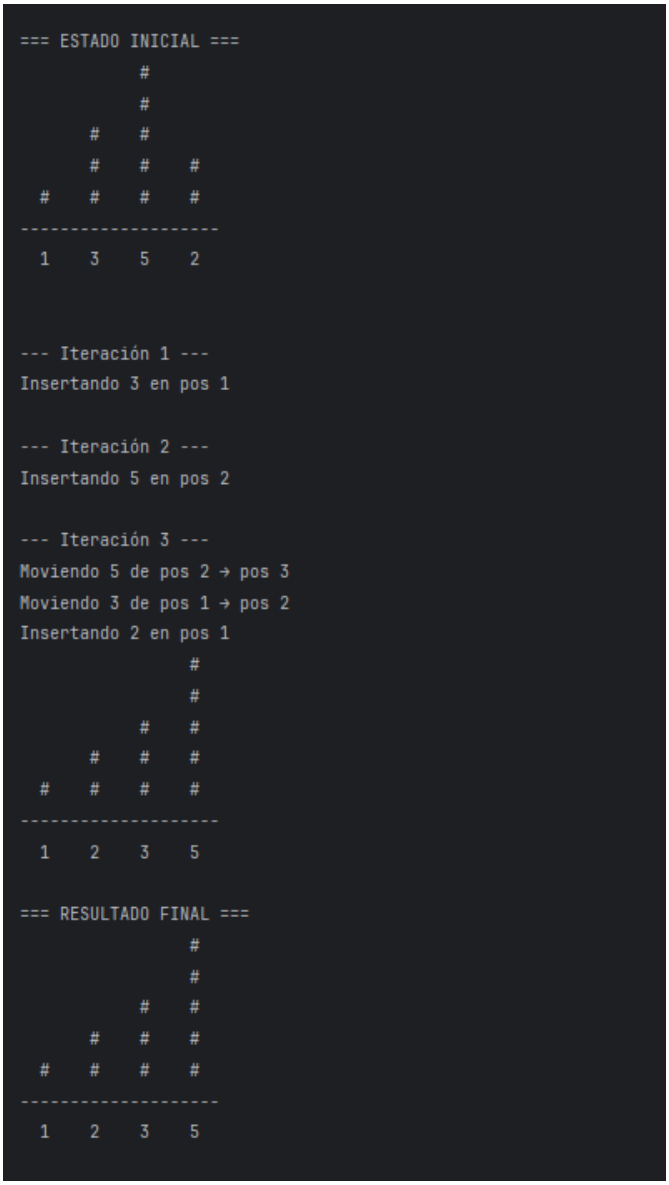
5. Resultados

Link del github https://github.com/FrancisValdiviezoUNL/Ordenamiento-Estructura.git

Documento txt de consola se encuentra en el repositorio de github.

- Capturas

- Inserción (1 3 5 2)



- Selección (1 3 5 2)

```
=== ESTADO INICIAL ===
           #
           #
      #    #
      #    #    #
  #   #    #    #
--------------------
  1    3    5    2

Swap 1: 3 ⇔ 2
           #
           #
           #    #
      #    #    #
  #   #    #    #
--------------------
  1    2    5    3

Swap 2: 5 ⇔ 3
                #
                #
           #    #
      #    #    #
  #   #    #    #
--------------------
  1    2    3    5

Swaps totales: 2
=== RESULTADO FINAL ===
                #
                #
           #    #
      #    #    #
  #   #    #    #
--------------------
  1    2    3    5
```

- Burbuja (1 3 5 2)

```
=== ESTADO INICIAL ===
          #
          #
      #   #
      #   #   #
  #   #   #   #
--------------------
  1   3   5   2


Paso 2:
          #
          #
          #   #
      #   #   #
  #   #   #   #
--------------------
  1   2   5   3


Paso 3:
              #
              #
          #   #
      #   #   #
  #   #   #   #
--------------------
  1   2   3   5

=== RESULTADO FINAL ===
              #
              #
          #   #
      #   #   #
  #   #   #   #
--------------------
  1   2   3   5
```

- Datos comparativos:

=== MENU DE ORDENAMIENTO ===
Ingrese números separados por espacio:
6 2 8 5 1

==========
RESUMEN FINAL
==========
Método | Tiempo (ms) | Resultado

Inserción | 1,2552 | [1, 2, 5, 6, 8]
Selección | 1,8111 | [1, 2, 5, 6, 8]

Burbuja | 1,4126 | [1, 2, 5, 6, 8]
---

6. Preguntas de Control

- **¿Por qué Inserción es preferible con datos casi ordenados?**

Porque Inserción aprovecha mucho cuando los datos ya vienen en con un buen orden. Solo necesita hacer unos pocos movimientos para dejar todo listo, así que termina rápido y sin complicarse asi que cuando la lista ya está casi ordenada, Inserción trabaja de manera muy eficiente

- **¿Qué propiedad hace que Selección use pocos swaps? ¿Qué compromisos tiene?**

El método de Selección usa pocos swaps porque en cada pasada solo hace un intercambio, y eso es cuando ya encontró el mínimo del resto de la lista. Esa es la razón por la que no mueve muchos elementos.

El compromiso es que, aunque casi no intercambia, tiene que revisar toda la lista una y otra vez.
Eso hace que el tiempo que tarda no mejore aunque los datos estén casi ordenados.

- **¿Cómo implementarías el corte temprano en Burbuja y qué caso reduce significativamente?**

Para implementar el corte temprano necesitas que detecte si en una pasada completa no hubo ningún intercambio.
Si eso pasa, significa que la lista ya está ordenada y el algoritmo puede detenerse de inmediato.

Este corte reduce muchísimo el tiempo cuando la lista ya está ordenada o casi ordenada, porque evita seguir revisando sin necesidad.

- **¿Cuál(es) de los tres puede(n) ser estable y en qué condiciones? Menciona dos casos borde que deben probarse siempre.**

Contamos con 3 métodos que podemos ser estables pero cumpliendo ciertas condiciones, a continuación se detalla cada uno de estos métodos:

- **Inserción** puede ser estable sin problemas, mientras se respete su forma normal de comparación.
- **Burbuja** también puede ser estable si solo intercambia cuando un elemento es realmente mayor al que sigue.
- **Selección**, en la mayoría de casos, no es estable porque puede mover un elemento igual desde una posición muy lejana.

También, tenemos dos casos borde que siempre deben probarse son:

1. **Arreglos con elementos duplicados**, para saber si se mantiene el orden original.
2. **Arreglos vacíos o de tamaño 1**, porque deben comportarse correctamente sin errores.

7. Conclusiones

En conclusión, tenemos que mediante este taller logramos aprender a como implementar 3 métodos de ordenamiento, pero con ventajas y desventajas de cada uno. Además de ver cual es el que menos interacciones realiza al momento de ordenar

8. Recomendaciones

Como recomendación tenemos que tenemos que probar con diferentes casos, tanto con datos bien desordenados, ordenados, y los diferentes posibilidades para ir comprobando su funcionamiento.