Reporte Técnico de Actividades Práctico-Experimentales Nro. 005

1. Datos de Identificación del Estudiante y la Práctica

Nombre del estudiante(s)	Royel Ivan Jima Pardo Darwin Ricardo Jimbo Jaramillo
Asignatura	Estructura de Datos
Ciclo	3 A
Unidad	2
Resultado de aprendizaje de la unidad	Aplica los métodos de ordenación y búsqueda en la resolución de problemas, bajo los principios transparencia, de solidaridad, responsabilidad honestidad.
Práctica Nro.	005
Título de la Práctica	Ordenación básica en Java: Burbuja, Selección e Inserción
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 13 de Noviembre Viernes 14 de Noviembre
Horario	07h30 – 10h30 07h30 – 09h30
Lugar	Aula
Tiempo planificado en el Sílabo	5 horas

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

- Guía de pruebas con datasets y salidas esperadas.

4. Equipos y Herramientas

- 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).

5. Procedimiento / Metodología Ejecutada

Para el desarrollo de esta práctica se siguió una metodología estructurada y progresiva, asegurando la correcta implementación de los algoritmos de ordenación, la arquitectura por capas y las funcionalidades adicionales solicitadas. A continuación, se detallan las etapas principales:

1. Definición del alcance y criterios de éxito

Se identificaron los requerimientos del Taller 5 y se definió como objetivo implementar tres algoritmos de ordenación (Inserción, Selección y Burbuja), con soporte para trazas visuales, lectura desde archivos `.txt`, entrada manual de datos y un menú interactivo. También se estableció la necesidad de construir una arquitectura por capas (domain, model, service, structure, exception, presentation) para mantener un nivel profesional en el código.

2. Diseño de la arquitectura y organización del proyecto

Se diseñó la estructura del proyecto siguiendo principios de separación de responsabilidades. Se creó un modelo de dominio para representar datasets y listas de números, un módulo de servicios para las implementaciones de los algoritmos, y componentes especializados para visualización ASCII, manejo de archivos y utilidades. Paralelamente, se configuró el entorno de trabajo, el repositorio Git y las carpetas del proyecto.

3. Implementación de los algoritmos de ordenación

Se desarrollaron los tres métodos principales de forma in-place y comparativa, respetando las propiedades teóricas de cada uno e incorporando comentarios detallados:

- Inserción (estable, eficiente para datos casi ordenados).
- Selección (minimiza swaps, no estable).
- Burbuja (estable, con optimización por corte temprano).

Cada algoritmo incluyó una sobrecarga con trazas (`sort(int[], boolean trace)`), permitiendo visualizar el proceso paso a paso.

4. Construcción de la visualización tipo VisuAlgo

Se implementó un módulo especializado (`ConsoleVisualizer`) encargado de mostrar el estado del arreglo en cada paso.
Este componente incorpora:

- Barras horizontales ASCII
- Colores ANSI para resaltar comparaciones, swaps y estados finales
- Animación con delay configurable (150 ms)
- Representación amigable para el usuario

Esta etapa requirió pruebas constantes para asegurar fluidez, claridad y compatibilidad entre sistemas operativos.

5. Desarrollo del menú interactivo y flujo de usuario

Se creó un módulo de presentación que organiza todas las opciones del sistema:

- Selección del algoritmo de ordenación
- Ingreso manual de valores
- Carga desde archivo `.txt`
- Uso de datasets predefinidos
- Activación/desactivación de trazas visuales
- Registro y visualización de historial

Se verificó que el flujo fuera claro, consistente y resistente a errores.

6. Manejo de archivos y excepciones personalizadas

Se implementó el lector de archivos (`FileUtils`) con validaciones para formatos incorrectos, líneas vacías o valores no numéricos.

Se diseñaron excepciones personalizadas para proporcionar mensajes claros y mejorar la robustez del sistema ante fallos del usuario o entradas inválidas.

7. Pruebas funcionales y validación de casos borde

Se realizaron pruebas de:

- Arreglos vacíos
- Arreglos con tamaño 1
- Datos ya ordenados
- Orden inverso
- Elementos duplicados
- Datasets A–E del taller

Se verificó el correcto funcionamiento de las trazas, el corte temprano en burbuja, la estabilidad cuando corresponde y la precisión de las métricas generadas (swaps, movimientos, tiempo).

8. Consolidación del historial y reportes

Se implementó un módulo de historial que almacena:

- Algoritmo utilizado
- Arreglo original y final
- Swaps, movimientos y comparaciones
- Tiempo de ejecución
- Fecha y hora

Esto permitió revisar la eficacia de cada método y documentar la experiencia del usuario.

9. Revisión final, ajustes y preparación para entrega

Finalmente, se realizó una revisión completa del código utilizando:

- Pruebas manuales
- Validación de arquitectura
- Revisión de comentarios Javadoc
- Limpieza de código

Se preparó el proyecto para su ejecución desde consola y se generó la documentación en Markdown para publicarse en GitHub y EVA.

6. Resultados

Dataset A

Inserción:

```
[ START INSERTION ]
  8: ********
  3: ***
  6: ******
  3: ***
  9: *********
```

```
[ END INSERTION ]
  3: ***
  3: ***
  6: ******
  8: ********
  9: *********

Resultado final: [3, 3, 6, 8, 9]
Métricas: swaps=0, moves=8 | Duración: 2128 ms
```

Selección:

```
[ START SELECTION ]
  8: ********
  3: ***
  6: ******
  3: ***
  9: *********
```

```
[ END SELECTION ]
  3: ***
  3: ***
  6: ******
  8: ********
  9: *********

Resultado final: [3, 3, 6, 8, 9]
Métricas: swaps=2, moves=0 | Duración: 2878 ms
```

Burbuja:

```
[ START BUBBLE ]
  8: ********
  3: ***
  6: ******
  3: ***
  9: *********
```

```
[ END BUBBLE ]
  3: ***
  3: ***
  6: ******
  8: ********
  9: *********

Resultado final: [3, 3, 6, 8, 9]
Métricas: swaps=4, moves=0 | Duración: 2721 ms
```

Dataset B

Inserción:

```
[ START INSERTION ]
  5: *****
  4: ****
  3: ***
  2: **
  1: *
```

```
[ END INSERTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=0, moves=14 | Duración: 3024 ms
```

Selección:

```
[ START SELECTION ]
  5: *****
  4: ****
  3: ***
  2: **
  1: *
```

```
[ END SELECTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=2, moves=0 | Duración: 3323 ms
```

Burbuja:

```
[ START BUBBLE ]
  5: *****
  4: ****
  3: ***
  2: **
  1: *
```

```
[ END BUBBLE ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=10, moves=0 | Duración: 3923 ms
```

Dataset C

Inserción:

```
[ START INSERTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****
```

```
[ END INSERTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=0, moves=4 | Duración: 1508 ms
```

Selección:

```
[ START SELECTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****
```

```
[ END SELECTION ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=0, moves=0 | Duración: 2417 ms
```

Burbuja:

```
[ START BUBBLE ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****
```

```
[ END BUBBLE ]
  1: *
  2: **
  3: ***
  4: ****
  5: *****

Resultado final: [1, 2, 3, 4, 5]
Métricas: swaps=0, moves=0 | Duración: 1054 ms
```

Dataset D

Inserción:

```
[ START INSERTION ]
  2: **
  2: **
  2: **
  2: **
```

```
[ END INSERTION ]
  2: **
  2: **
  2: **
  2: **

Resultado final: [2, 2, 2, 2]
Métricas: swaps=0, moves=3 | Duración: 1210 ms
```

Selección:

```
[ START SELECTION ]
  2: **
  2: **
  2: **
  2: **
```

```
[ END SELECTION ]
  2: **
  2: **
  2: **
  2: **

Resultado final: [2, 2, 2, 2]
Métricas: swaps=0, moves=0 | Duración: 1658 ms
```

Burbuja:

```
[ START BUBBLE ]
  2: **
  2: **
  2: **
  2: **
```

```
[ END BUBBLE ]
  2: **
  2: **
  2: **
  2: **

Resultado final: [2, 2, 2, 2]
Métricas: swaps=0, moves=0 | Duración: 904 ms
```

Dataset E

Inserción:

```
[ START INSERTION ]
  9: *********
  1: *
  8: ********
  2: **
```

```
[ END INSERTION ]
  1: *
  2: **
  8: ********
  9: *********

Resultado final: [1, 2, 8, 9]
Métricas: swaps=0, moves=7 | Duración: 1810 ms
```

Selección:

```
[ START SELECTION ]
  9: *********
  1: *
  8: ********
  2: **
```

```
[ END SELECTION ]
  1: *
  2: **
  8: ********
  9: *********

Resultado final: [1, 2, 8, 9]
Métricas: swaps=2, moves=0 | Duración: 2107 ms
```

Burbuja:

```
[ START BUBBLE ]
  9: *********
  1: *
  8: ********
  2: **
```

```
[ END BUBBLE ]
  1: *
  2: **
  8: ********
  9: *********

Resultado final: [1, 2, 8, 9]
Métricas: swaps=4, moves=0 | Duración: 2263 ms
```

- **Tabla Comparativa**

Algoritmo	**Ventajas**	**Desventajas**
Inserccion	estable, rapido para casi ordenados	lento para casos grandes
Seleccion	Muy pocos swaps	No estable, muchas comparaciones
Burbuja	Simple, visual	Lento sin optimización

7. Preguntas de Control

1-. ¿Por qué la Inserción es preferible con datos casi ordenados?

Porque su complejidad se reduce a O(n) cuando los elementos están casi en su posición correcta, ya que realiza pocos desplazamientos y no muchas comparaciones.

2-. ¿Qué propiedad hace que la Selección use pocos swaps? ¿Qué compromisos tiene?

Usa pocos swaps porque solo intercambia una vez por iteración externa, al colocar el mínimo en la posición correcta. Uno de sus compromisos es que realiza muchas comparaciones ($O(n^2)$) incluso si el arreglo ya está ordenado.

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

Se revisa si en una pasada no ocurre ningún swap, si no hubo intercambios, el algoritmo finaliza. Se reduciría significativamente el tiempo cuando el arreglo está ya ordenado o casi ordenado.

4-. ¿Cuál(es) de los tres puede(n) ser estable y en qué condiciones?

InsertionSort: es estable si la comparación permite mantener el orden relativo.

BubbleSort: es estable si solo intercambia cuando el elemento izquierdo es estrictamente mayor.

SelectionSort: normalmente no es estable porque puede mover un elemento mínimo saltando otros iguales.

5-. Menciona dos casos borde que deben probarse siempre.

1) Arreglo vacío.
2) Arreglo de tamaño 1.

8. Conclusiones

- La práctica permitió comprender el funcionamiento de los algoritmos clásicos de ordenación Inserción, Selección y Burbuja desde una perspectiva teórica, práctica y visual. A través de la implementación in-place y comparativa de cada método, fue posible analizar su comportamiento en términos de estabilidad, uso de memoria, cantidad de intercambios y eficacia frente a distintos tipos de arreglos, incluidos los casos borde del taller.
- La incorporación de un visualizador ASCII estilo VisuAlgo facilitó el entendimiento del proceso interno de ordenación, destacando comparaciones, swaps y movimientos mediante barras, colores y animación por pasos. Además, la arquitectura por capas implementada aportó solidez, escalabilidad y claridad al proyecto, dejando una base profesional que facilita futuras ampliaciones, como nuevos algoritmos, análisis de complejidad o integración de archivos externos.

9. Recomendaciones

- InsertionSort es recomendable cuando los datos están casi ordenados, el tamaño del arreglo es pequeño o se requiere estabilidad, ya que en estos escenarios ofrece un rendimiento muy eficiente y un comportamiento fácil de depurar mediante trazas visuales.

- SelectionSort resulta adecuado cuando es necesario minimizar la cantidad de intercambios o cuando el costo del swap es elevado. Es útil con fines didácticos para entender el proceso de búsqueda del mínimo, aunque no es estable y puede resultar menos eficiente en comparaciones.

- BubbleSort es ideal para fines educativos por su claridad visual, especialmente con la implementación de corte temprano, que mejora su rendimiento en datos casi ordenados. Aunque no es eficiente para arreglos grandes, es muy útil para visualizar paso a paso el proceso de intercambio de elementos.

10. Bibliografía / Referencias

[1] J. A. Ripoll, «Algoritmos básicos: algoritmos de ordenación», 14 de febrero de 2025. https://www.juanantonioripoll.es/metodologia-de-programacion-python/algoritmos-basicos-algoritmos-de-ordenacion.aspx

[2] Elbloddeldev, «Algoritmos de Ordenamiento: Ejemplos en JavaScript, Python, Java y C++», *El Blog del Programador*, 17 de noviembre de 2025. [En línea]. Disponible en: https://elblogdelprogramador.com/posts/algoritmos-de-ordenamiento-ejemplos-en-javascript-python-java-y-c/#gsc.tab=0

11. Anexos

- Sorting (Bubble, Selection, Insertion, Merge, Quick, Counting, Radix) - VisuAlgo

- Link de GitHub: https://github.com/R0yalCode/Ordenacion-en-Java.git