Reporte Técnico de Actividades Práctico-Experimentales Nro. 006

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.	006
Título de la Práctica	Ordenación básica en Java: Burbuja, Selección e Inserción
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 20 de noviembre Viernes 21 de noviembre
Horario	07h30 – 10h30 07h30 – 09h30
Lugar	Aula
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

- Datasets CSV (citas, pacientes e inventario).
- Guía de pruebas entregada en el taller.

4. Equipos y Herramientas

- **OpenJDK 21 / 25**
- **IDE**: VSCode / IntelliJ
- **Git + GitHub**
- **EVA/Moodle** para entrega
- **Documentación**: Markdown + PDF
- **Generador de datasets reproducibles** (semilla 42)

5. Procedimiento / Metodología Ejecutada

La metodología aplicada corresponde al enfoque ABPr descrito en el taller, y fue adaptada profesionalmente al proyecto real implementado.

1. Generación, validación y preparación de los cuatro datasets CSV

Antes de implementar los algoritmos o el sistema comparador, se generaron los **cuatro datasets oficiales** solicitados en el taller:

1. **citas_100.csv** — datos aleatorios
2. **citas_100_casi_ordenadas.csv** — datos casi ordenados
3. **pacientes_500.csv** — datos aleatorios con duplicados
4. **inventario_500_inverso.csv** — orden totalmente inverso

Metodología aplicada para su construcción:

- Se utilizó una **semilla fija (42)** para asegurar reproducibilidad en cada ejecución.
- Se generaron **IDs únicos** con prefijo asociado al dataset.
- Se construyeron campos realistas: nombres, apellidos, fechas, horas, priorizaciones, inventario, etc.
- El formato fue estrictamente:
    - **CSV**
    - **delimitado con punto y coma “;”**
    - **codificación UTF-8 sin BOM**
    - encabezado en la primera fila
    - fechas en formato **ISO-8601 (YYYY-MM-DDTHH:MM)**.

Validaciones realizadas:

- Revisión manual de estructura y delimitadores.
- Apertura en VSCode y LibreOffice para verificar codificación UTF-8.
- Validación automática mediante el módulo `CsvReader`.
- Garantizar ausencia de filas vacías, caracteres ilegales o columnas inconsistentes.
- Inclusión final en la carpeta **`src/main/resources/data/`** para acceso mediante `getResourceAsStream()`.

Con este paso, los datasets quedaron listos para ser consumidos por el sistema comparador sin errores de formato, garantizando condiciones experimentales homogéneas.

2. Instrumentación

Se añadieron contadores en cada algoritmo:

- `comparisons++`
- `swaps++`

Además:

- Se midió tiempo con **System.nanoTime()**
- Se ejecutaron **R = 10 repeticiones**
- Se descartaron las **primeras 3 corridas** (calentamiento JVM/JIT)
- Se tomó la **mediana de las 7 restantes**.
- Se aisló completamente la carga del CSV fuera de la medición.

Esta instrumentación está implementada en `SortService`.

3. Casos de prueba

Los datasets del proyecto:

Archivo CSV	**n**	**Tipo**	**Clave**
citas_100.csv	100	aleatorio	fechaHora
citas_100_casi_ordenadas.csv	100	casi ordenado	fechaHora
pacientes_500.csv	500	aleatorio con duplicados	apellido
inventario_500_inverso.csv	500	inverso	stock

Procedimiento:

- Se definió la clave de orden para cada dataset.

- El sistema convierte los registros a objetos del dominio (`Cita`, `Paciente`, `Inventario`).
- Cada objeto implementa `Comparable`.
- Se evalúan los algoritmos:
    - Inserción
    - Selección
    - Burbuja (con corte temprano)

El sistema registra:
n, tipo, comparaciones, swaps, tiempo mediana.

4. Análisis

Para cada dataset se comparó:

- comportamiento en datos casi ordenados
- impacto de duplicados
- penalización en orden inverso
- estabilidad, swaps y tiempo
- gráfica conceptual tiempo vs. tipo de orden

Se construyó una matriz de recomendación práctica (**ver sección 7**).

5. Historial y reportes

El proyecto genera automáticamente un historial con:

- algoritmo utilizado
- dataset
- comparaciones
- swaps
- tiempo
- fecha/hora
- mediana final

Este historial puede exportarse a TXT.

6. 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 1: **citas_100.csv** (aleatorio, n=100)

```
Selecciona una opción: 1
Introduce la ruta completa del CSV: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\citas_100.csv
Archivo seleccionado: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\citas_100.csv
1) Evaluar los 3 algoritmos
2) Evaluar con Inserción
3) Evaluar con Selección
4) Evaluar con Burbuja
Selecciona: 1
Registros cargados: 100

=== Ejecutando: Inserción ===
Corrida 1: tiempo(ns)=2061300 comparisons=2245 swaps=2148
Corrida 2: tiempo(ns)=297400 comparisons=2245 swaps=2148
Corrida 3: tiempo(ns)=266400 comparisons=2245 swaps=2148
Corrida 4: tiempo(ns)=255300 comparisons=2245 swaps=2148
Corrida 5: tiempo(ns)=237100 comparisons=2245 swaps=2148
Corrida 6: tiempo(ns)=239800 comparisons=2245 swaps=2148
Corrida 7: tiempo(ns)=240000 comparisons=2245 swaps=2148
Corrida 8: tiempo(ns)=195500 comparisons=2245 swaps=2148
Corrida 9: tiempo(ns)=205100 comparisons=2245 swaps=2148
Corrida 10: tiempo(ns)=453200 comparisons=2245 swaps=2148
Resultado Inserción: n=100  tiempo_mediana(ns)=239800  comparisons_mediana=2245  swaps_mediana=2148
```

```
=== Ejecutando: Selección ===
Corrida 1: tiempo(ns)=357900 comparisons=4950 swaps=94
Corrida 2: tiempo(ns)=309600 comparisons=4950 swaps=94
Corrida 3: tiempo(ns)=313600 comparisons=4950 swaps=94
Corrida 4: tiempo(ns)=322600 comparisons=4950 swaps=94
Corrida 5: tiempo(ns)=300700 comparisons=4950 swaps=94
Corrida 6: tiempo(ns)=300300 comparisons=4950 swaps=94
Corrida 7: tiempo(ns)=331200 comparisons=4950 swaps=94
Corrida 8: tiempo(ns)=304400 comparisons=4950 swaps=94
Corrida 9: tiempo(ns)=323100 comparisons=4950 swaps=94
Corrida 10: tiempo(ns)=368700 comparisons=4950 swaps=94
Resultado Selección: n=100  tiempo_mediana(ns)=322600  comparisons_mediana=4950  swaps_mediana=94
```

```
=== Ejecutando: Burbuja ===
Corrida 1: tiempo(ns)=459900 comparisons=4922 swaps=2148
Corrida 2: tiempo(ns)=425600 comparisons=4922 swaps=2148
Corrida 3: tiempo(ns)=479500 comparisons=4922 swaps=2148
Corrida 4: tiempo(ns)=550300 comparisons=4922 swaps=2148
Corrida 5: tiempo(ns)=515800 comparisons=4922 swaps=2148
Corrida 6: tiempo(ns)=535500 comparisons=4922 swaps=2148
Corrida 7: tiempo(ns)=533500 comparisons=4922 swaps=2148
Corrida 8: tiempo(ns)=2409800 comparisons=4922 swaps=2148
Corrida 9: tiempo(ns)=489600 comparisons=4922 swaps=2148
Corrida 10: tiempo(ns)=503500 comparisons=4922 swaps=2148
Resultado Burbuja: n=100  tiempo_mediana(ns)=533500  comparisons_mediana=4922  swaps_mediana=2148
```

Algoritmo	Comparación	Swaps	Tiempo mediana (ns)
Inserción	2245	2148	239000
Selección	4950	94	322600
Burbuja	4922	2148	533500

Inserción supera claramente a Burbuja y Selección en aleatorios pequeños.

- Dataset 2: **citas_100_casi_ordenadas.csv** (casi ordenado)

```
Selecciona una opción: 1
Introduce la ruta completa del CSV: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\citas_100_casi_ordenadas.csv
Archivo seleccionado: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\citas_100_casi_ordenadas.csv
1) Evaluar los 3 algoritmos
2) Evaluar con Inserción
3) Evaluar con Selección
4) Evaluar con Burbuja
Selecciona: 1
Registros cargados: 100

=== Ejecutando: Inserción ===
Corrida 1: tiempo(ns)=980400 comparisons=342 swaps=243
Corrida 2: tiempo(ns)=110600 comparisons=342 swaps=243
Corrida 3: tiempo(ns)=74800 comparisons=342 swaps=243
Corrida 4: tiempo(ns)=80900 comparisons=342 swaps=243
Corrida 5: tiempo(ns)=84400 comparisons=342 swaps=243
Corrida 6: tiempo(ns)=164100 comparisons=342 swaps=243
Corrida 7: tiempo(ns)=157100 comparisons=342 swaps=243
Corrida 8: tiempo(ns)=83600 comparisons=342 swaps=243
Corrida 9: tiempo(ns)=69200 comparisons=342 swaps=243
Corrida 10: tiempo(ns)=53800 comparisons=342 swaps=243
Resultado Inserción: n=100  tiempo_mediana(ns)=83600  comparisons_mediana=342  swaps_mediana=243
```

```
=== Ejecutando: Selección ===
Corrida 1: tiempo(ns)=428400 comparisons=4950 swaps=5
Corrida 2: tiempo(ns)=445800 comparisons=4950 swaps=5
Corrida 3: tiempo(ns)=389300 comparisons=4950 swaps=5
Corrida 4: tiempo(ns)=296000 comparisons=4950 swaps=5
Corrida 5: tiempo(ns)=271100 comparisons=4950 swaps=5
Corrida 6: tiempo(ns)=271500 comparisons=4950 swaps=5
Corrida 7: tiempo(ns)=271200 comparisons=4950 swaps=5
Corrida 8: tiempo(ns)=281400 comparisons=4950 swaps=5
Corrida 9: tiempo(ns)=305600 comparisons=4950 swaps=5
Corrida 10: tiempo(ns)=276600 comparisons=4950 swaps=5
Resultado Selección: n=100  tiempo_mediana(ns)=276600  comparisons_mediana=4950  swaps_mediana=5
```

```
=== Ejecutando: Burbuja ===
Corrida 1: tiempo(ns)=358100 comparisons=4170 swaps=243
Corrida 2: tiempo(ns)=486800 comparisons=4170 swaps=243
Corrida 3: tiempo(ns)=339600 comparisons=4170 swaps=243
Corrida 4: tiempo(ns)=280700 comparisons=4170 swaps=243
Corrida 5: tiempo(ns)=303000 comparisons=4170 swaps=243
Corrida 6: tiempo(ns)=269700 comparisons=4170 swaps=243
Corrida 7: tiempo(ns)=301700 comparisons=4170 swaps=243
Corrida 8: tiempo(ns)=288700 comparisons=4170 swaps=243
Corrida 9: tiempo(ns)=284500 comparisons=4170 swaps=243
Corrida 10: tiempo(ns)=304900 comparisons=4170 swaps=243
Resultado Burbuja: n=100  tiempo_mediana(ns)=288700  comparisons_mediana=4170  swaps_mediana=243
```

Algoritmo	**Comparación**	**Swaps**	**Tiempo mediana (ns)**
Inserción	342	243	83600
Selección	4950	5	276600
Burbuja	4170	243	288700

Para este caso Inserción y **Burbuja** son los más factibles a utilizar.

- Dataset 3: **pacientes_500.csv** (aleatorio con duplicados)

```
Selecciona una opción: 1
Introduce la ruta completa del CSV: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\pacientes_500.csv
Archivo seleccionado: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\pacientes_500.csv
1) Evaluar los 3 algoritmos
2) Evaluar con Inserción
3) Evaluar con Selección
4) Evaluar con Burbuja
Selecciona: 1
Registros cargados: 500

=== Ejecutando: Inserción ===
Corrida 1: tiempo(ns)=12195600 comparisons=58042 swaps=57545
Corrida 2: tiempo(ns)=1538500 comparisons=58042 swaps=57545
Corrida 3: tiempo(ns)=987600 comparisons=58042 swaps=57545
Corrida 4: tiempo(ns)=1081800 comparisons=58042 swaps=57545
Corrida 5: tiempo(ns)=979600 comparisons=58042 swaps=57545
Corrida 6: tiempo(ns)=1002300 comparisons=58042 swaps=57545
Corrida 7: tiempo(ns)=963300 comparisons=58042 swaps=57545
Corrida 8: tiempo(ns)=967200 comparisons=58042 swaps=57545
Corrida 9: tiempo(ns)=1412400 comparisons=58042 swaps=57545
Corrida 10: tiempo(ns)=1022700 comparisons=58042 swaps=57545
Resultado Inserción: n=500  tiempo_mediana(ns)=1002300  comparisons_mediana=58042  swaps_mediana=57545
```

```
=== Ejecutando: Selección ===
Corrida 1: tiempo(ns)=2899400 comparisons=124750 swaps=463
Corrida 2: tiempo(ns)=1788100 comparisons=124750 swaps=463
Corrida 3: tiempo(ns)=1637200 comparisons=124750 swaps=463
Corrida 4: tiempo(ns)=1683000 comparisons=124750 swaps=463
Corrida 5: tiempo(ns)=2324000 comparisons=124750 swaps=463
Corrida 6: tiempo(ns)=2405200 comparisons=124750 swaps=463
Corrida 7: tiempo(ns)=1632600 comparisons=124750 swaps=463
Corrida 8: tiempo(ns)=1954100 comparisons=124750 swaps=463
Corrida 9: tiempo(ns)=2242600 comparisons=124750 swaps=463
Corrida 10: tiempo(ns)=2334700 comparisons=124750 swaps=463
Resultado Selección: n=500  tiempo_mediana(ns)=2242600  comparisons_mediana=124750  swaps_mediana=463
```

```
=== Ejecutando: Burbuja ===
Corrida 1: tiempo(ns)=4838100 comparisons=124579 swaps=57545
Corrida 2: tiempo(ns)=2707400 comparisons=124579 swaps=57545
Corrida 3: tiempo(ns)=3377900 comparisons=124579 swaps=57545
Corrida 4: tiempo(ns)=3829800 comparisons=124579 swaps=57545
Corrida 5: tiempo(ns)=3124000 comparisons=124579 swaps=57545
Corrida 6: tiempo(ns)=2947200 comparisons=124579 swaps=57545
Corrida 7: tiempo(ns)=2674000 comparisons=124579 swaps=57545
Corrida 8: tiempo(ns)=2340600 comparisons=124579 swaps=57545
Corrida 9: tiempo(ns)=1941900 comparisons=124579 swaps=57545
Corrida 10: tiempo(ns)=2710100 comparisons=124579 swaps=57545
Resultado Burbuja: n=500  tiempo_mediana(ns)=2710100  comparisons_mediana=124579  swaps_mediana=57545
```

Algoritmo	**Comparación**	**Swaps**	**Tiempo mediana (ns)**
Inserción	5842	57545	1002300
Selección	124750	463	2242600
Burbuja	124579	57545	2710100

Inserción se comporta mejor porque mantiene estabilidad útil en duplicados.

- Dataset 4: **inventario_500_inverso.csv** (inverso total)

```
Selecciona una opción: 1
Introduce la ruta completa del CSV: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\inventario_500_inverso.csv
Archivo seleccionado: C:\Users\ivanj\Documents\UNL\CICLO 3\ESTRUTURA DE DATOS\ordenacion\ordenacion\ordenacion\resources\inventario_500_inverso.csv
1) Evaluar los 3 algoritmos
2) Evaluar con Inserción
3) Evaluar con Selección
4) Evaluar con Burbuja
Selecciona: 1
Registros cargados: 500

=== Ejecutando: Inserción ===
Corrida 1: tiempo(ns)=7285100 comparisons=124750 swaps=124750
Corrida 2: tiempo(ns)=2975100 comparisons=124750 swaps=124750
Corrida 3: tiempo(ns)=1803300 comparisons=124750 swaps=124750
Corrida 4: tiempo(ns)=2767900 comparisons=124750 swaps=124750
Corrida 5: tiempo(ns)=1821700 comparisons=124750 swaps=124750
Corrida 6: tiempo(ns)=2124300 comparisons=124750 swaps=124750
Corrida 7: tiempo(ns)=5662700 comparisons=124750 swaps=124750
Corrida 8: tiempo(ns)=2771500 comparisons=124750 swaps=124750
Corrida 9: tiempo(ns)=1841600 comparisons=124750 swaps=124750
Corrida 10: tiempo(ns)=2372400 comparisons=124750 swaps=124750
Resultado Inserción: n=500  tiempo_mediana(ns)=2372400  comparisons_mediana=124750  swaps_mediana=124750
```

```
=== Ejecutando: Selección ===
Corrida 1: tiempo(ns)=6928500 comparisons=124750 swaps=250
Corrida 2: tiempo(ns)=3618700 comparisons=124750 swaps=250
Corrida 3: tiempo(ns)=435000 comparisons=124750 swaps=250
Corrida 4: tiempo(ns)=530100 comparisons=124750 swaps=250
Corrida 5: tiempo(ns)=372200 comparisons=124750 swaps=250
Corrida 6: tiempo(ns)=373100 comparisons=124750 swaps=250
Corrida 7: tiempo(ns)=365500 comparisons=124750 swaps=250
Corrida 8: tiempo(ns)=495200 comparisons=124750 swaps=250
Corrida 9: tiempo(ns)=362900 comparisons=124750 swaps=250
Corrida 10: tiempo(ns)=372300 comparisons=124750 swaps=250
Resultado Selección: n=500  tiempo_mediana(ns)=372300  comparisons_mediana=124750  swaps_mediana=250
```

```
=== Ejecutando: Burbuja ===
Corrida 1: tiempo(ns)=12187900 comparisons=124750 swaps=124750
Corrida 2: tiempo(ns)=2813100 comparisons=124750 swaps=124750
Corrida 3: tiempo(ns)=3074400 comparisons=124750 swaps=124750
Corrida 4: tiempo(ns)=1681300 comparisons=124750 swaps=124750
Corrida 5: tiempo(ns)=2673000 comparisons=124750 swaps=124750
Corrida 6: tiempo(ns)=1651300 comparisons=124750 swaps=124750
Corrida 7: tiempo(ns)=1605400 comparisons=124750 swaps=124750
Corrida 8: tiempo(ns)=2876200 comparisons=124750 swaps=124750
Corrida 9: tiempo(ns)=1540100 comparisons=124750 swaps=124750
Corrida 10: tiempo(ns)=1597500 comparisons=124750 swaps=124750
Resultado Burbuja: n=500  tiempo_mediana(ns)=1651300  comparisons_mediana=124750  swaps_mediana=124750
```

Algoritmo	**Comparación**	**Swaps**	**Tiempo mediana (ns)**
Inserción	12475	12475	2372400
Selección	12475	250	372300
Burbuja	124750	123750	1651300

Selección es menos penalizado que Inserción y Burbuja.

7. Matriz de Recomendación

Condición	**Algoritmo recomendado**	**Justificación**
Datos casi ordenados	**Inserción** o **Burbuja con corte temprano**	Menos desplazamientos; termina antes
Minimizar swaps	Selección	Realiza ~n swaps en total
Muchos duplicados	Inserción	Mantiene estabilidad
Arreglo pequeño	Inserción	Excelente rendimiento
Arreglo inverso	Selección	Comparaciones constantes
Visualización educativa	Burbuja	Más intuitivo

8. Preguntas de Control

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

Porque la salida por consola es una operación lenta, bloqueante y ajena al algoritmo.

Escribir texto en pantalla introduce:

- latencia del sistema operativo
- tiempos de I/O
- variación por búfer
- interrupciones de CPU

Esto altera completamente la medición del ordenamiento, por lo que el tiempo ya no representa el tiempo real del algoritmo sino el del sistema de salida estándar.

2-. Explica por qué Selección tiene comparaciones ~ n(n−1)/2 independientemente del orden

Porque el algoritmo recorre toda la parte no ordenada del arreglo para encontrar el mínimo en cada iteración externa.

No importa si el arreglo ya está ordenado, es casi ordenado o inverso: siempre compara todos los pares posibles, ejecutando exactamente:

$$(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$$

Incluso cuando detecta el mínimo antes, igual debe revisar el resto para confirmarlo.

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

Porque Inserción desplaza hacia la izquierda solo mientras el elemento “actual” sea menor que el anterior.

En datos casi ordenados:

- prácticamente no realiza desplazamientos
- muy pocas comparaciones fallan
- la complejidad se aproxima a O(n)

Por eso es el algoritmo más rápido para casos casi ordenados.

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

En presencia de duplicados:

- Inserción conserva el orden relativo original → estable.
- Burbuja también puede ser estable si solo intercambia cuando a[j] > a[j+1].
- Selección casi siempre rompe estabilidad al mover el mínimo.

Los duplicados vuelven relevante la estabilidad porque modifican el orden lógico de los registros si no se conserva su posición relativa original.

5-. ¿Por qué Burbuja con corte temprano mejora en "casi ordenado" pero no en "inverso"?

En un arreglo casi ordenado:

- en la primera pasada no hay swaps
- entonces el algoritmo detecta "no hubo intercambios"
- y termina anticipadamente

Pero en un arreglo inverso:

- cada elemento está en la peor posición
- se realizan swaps en todas las pasadas
- nunca se activa el corte temprano
- se ejecutan las n−1 pasadas completas

Por eso Burbuja sigue siendo muy lento en inversos.

9. Conclusiones

La práctica permitió comprender con rigor el comportamiento real de los algoritmos Inserción, Selección y Burbuja, evaluados con datasets reales y mediciones profesionales (mediana, repetición R=10, aislamiento IO).

Los resultados muestran que:

- Inserción domina en casi ordenados y arreglos pequeños.
- Selección es predecible y con pocos swaps, útil cuando el costo de intercambio es alto.
- Burbuja es ineficiente para grandes n, salvo con corte temprano y datos casi ordenados.

La arquitectura por capas implementada permitió una separación profesional del dominio, servicios, infraestructura y presentación, facilitando mantenibilidad, extensibilidad y pruebas.

10. Recomendaciones

- Usar **Inserción** cuando se requiera estabilidad y rapidez en arreglos pequeños o casi ordenados.
- Usar **Selección** cuando se necesite reducir swaps independientemente del orden inicial.
- Usar **Burbuja** con corte temprano únicamente con fines educativos o visuales.
- Para análisis más avanzados incorporar JMH o amortiguar variabilidad JVM mediante mayor R.
- Mantener separación de IO y lógica de negocio para asegurar mediciones fiables.

11. Bibliografía / Referencias

[1] S. Gautam, «Bubble Sort, Selection Sort and Insertion Sort Algorithm». https://www.enjoyalgorithms.com/blog/introduction-to-sorting-bubble-sort-selection-sort-and-insertion-sort

12. Anexos

- Link de GitHub: https://github.com/R0yalCode/Comparacion-.git