Reporte Técnico de Actividades Práctico-Experimentales Nro. 006

1. Datos de Identificación del Estudiante y la Práctica

Nombre del estudiante(s)	- Soledad Buri - Gyna Yupanqui
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 de solidaridad, transparencia, responsabilidad y honestidad.
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

- 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

- 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

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos).

Inicio

- Presentación del objetivo comparativo y criterios de éxito.
- Formación de equipos (3–4) y revisión de la rúbrica.
- Creación de repo Git.
- Lineamientos de uso responsable de IA.

Desarrollo

- **Paso 1. Instrumentación (obligatorio)**
  - Añade contadores a tus algoritmos:
    - comparisons++ al comparar dos claves,
    - swaps++ al intercambiar posiciones.
  - Mide tiempo con `System.nanoTime()` sin imprimir durante la medición (las trazas distorsionan).
  - Ejecuta R repeticiones por caso (Xsug.: R=10), descarta las 3 primeras (calentamiento/JIT) y reporta la mediana de tiempo.
  - Aísla IO: carga CSV fuera de la medición; mide sólo el ordenamiento del array en memoria.

- **Paso 2. Casos de prueba**
  - Define clave de orden (p. ej., `fechaHora` en `citas`, `apellido` en `pacientes`, `stock` en `inventario`).
    - Convierte a array de la clave (o a registros con `Comparable` por clave).
    - Ejecuta: Insertion, Selection, Bubble (con "corte temprano" en

Burbuja).

- Registra: n, %casi-ordenado, %duplicados, comparisons, swaps, tiempo(ns) (mediana de R-3 corridas).

- **Paso 3. Análisis**
  - Tablas comparativas por caso (n, orden, duplicados) y gráficos (tiempo vs. n; tiempo vs. %casi-ordenado).
  - Matriz de recomendación (reglas prácticas):
  - Casi ordenado + n pequeño/medio → Inserción gana (menos
  - movimientos).
  - Muchos duplicados → Inserción tiende a mantener estabilidad útil; Selección hace n(n−1)/2 comparaciones siempre, con pocos swaps.
  - Inverso o aleatorio (n pequeño/educativo) → cualquiera, pero Burbuja penaliza; Selección constante en comparaciones; Inserción peor en inverso pero mejor si detecta localmente orden.

Cierre

Discusión guiada: ¿Cuándo conviene cada uno? ¿Qué sesgos introdujo la medición?

Algoritmo	Cuando conviene
Insertion Sort	Datasets pequeños o medianos, especialmente si están casi ordenados
Bubble Sort	Solo para listas pequeñas y como referencia académica
Selection Sort	Datasets desordenados de tamaño medio donde importar reducir swaps

Sesgos introducidos en la medición

Patrones del dataset

- Los CSV usados (ordenado, casi ordenado, invertido) favorecen a Insertion Sort y perjudican a Bubble y Selection.

Operación extra de copiar los arreglos

- La copia de datos antes de cada ejecución añade tiempo extra que afecta más a algoritmos rápidos.

Medición con objetos

- Comparar objetos (Comparable) es más lento y perjudica especialmente a Selection Sort.

Completar README e informe con evidencias y la matriz de recomendación.

- https://github.com/Zoe777zz/TALLER_06

6. Resultados

Leyenda de columnas

columna	significado
tiempo(ns)	duracion total del algoritmo
comparac	total de comparaciones realizadas
swaps	total de intercambios ejecutados
status	estado final del algoritmo

Dataset Citas 100 datos

algoritmo	tiempo(ns)	comparac	swaps	status
bubble	1965300	4950	2430	ok
insertion	526600	2526	2430	ok
selection	1123400	4950	94	ok

Dataset Citas 100 datos casi ordenadas

algoritmo	tiempo(ns)	comparac	swaps	status
bubble	422100	4797	341	ok
insertion	91700	440	341	ok
selection	681500	4950	5	ok

Dataset Pacientes 500 datos

algoritmo	tiempo(ns)	comparac	swaps	status
bubble	13815500	124084	60261	ok
insertion	3051300	60757	60261	ok
selection	5841300	124750	485	ok

Dataset Inventario 500 datos inverso

algoritmo	tiempo(ns)	comparac	swaps	status
bubble	7769200	124750	124750	ok
insertion	26997200	124750	124750	ok
selection	18671600	124750	250	ok

Discusión

Después de analizar juntas los resultados, concluimos que el rendimiento de cada algoritmo va a depender de las características de los datos. Insertion es más eficiente con los datasets casi ordenados reduciendo los movimientos y el tiempo de ejecución. Selection se mantuvo estable con todos los datasets probados realizando el mismo número de ejecuciones y Bubble mostro una ventaja con datos ya casi ordenados. Culminando con que depende mucho de los tipos de datos que se usen escoger el método que más convenga.

7. Preguntas de Control

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

  Porque las operaciones de salida a consola son mucho más lentas que las operaciones en memoria, y al imprimir constantemente se añade un retraso artificial que no refleja el verdadero tiempo del algoritmo.

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

  Porque este algoritmo siempre busca el mínimo recorriendo en cada iteración sin aprovechar ningún orden previo, por ello se realizan la misma cantidad de comparaciones sin importar de como estén los datos.

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

  Porque en este tipo de datos se desplazan muy poco por lo que el numero de comparaciones y movimientos disminuye.

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

Juegan un papel importante al conservar el orden original dentro de los algoritmos evitando reordenamientos innecesarios.

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

  Porque el corte temprano detecta rápidamente que no hubo intercambios y termina antes en arreglos casi ordenados. En cambio, en un arreglo inverso siempre habrá intercambios en cada pasada, por lo que no se puede detener antes.

8. Conclusiones

- Realizar pruebas con datasets se logró tener una mejor apreciación sobre el funcionamiento de los algoritmos y visualizar cual actúa mejor.
- De los tres algoritmos utilizados en el presente taller Insertion es la mejor opción para arreglos pequeños o casi ordenados.
- Se observo que para arreglos demasiado grandes y sin un orden previo ninguna de las tres opciones de algoritmos es óptimo.
- La estabilidad dentro de los algoritmos tiene un papel importante cuando existe duplicados en los datos en el caso de Insertion y Bubble son estables preservando el orden relativo, mientas que en Selection al no ser estable altera la posición original de los duplicados.

9. Recomendaciones

- Realizar pruebas con datasets muy grandes los algoritmos de Bubble y Selection se vuelven un poco ineficientes por la robustez de los datos.
- Cuando los datos contienen duplicados es recomendable utilizar los algoritmos estables como Isertion y Bubble.
- El algoritmo Selection al no ser estable se debe de evitar utilizarlo en aplicaciones donde existan datos con duplicados.

10. Bibliografía / Referencias

[1] "Insertion sort (with code in Python/C++/Java/C)", *Programiz.com*. [En línea]. Disponible en: https://www.programiz.com/dsa/insertion-sort. [Consultado: 21-nov-2025].

[2] "Bubble sort", *Programiz.com*. [En línea]. Disponible en: https://www.programiz.com/dsa/bubble-sort. [Consultado: 21-nov-2025].

[3] "Selection sort (with code in Python/C++/Java/C)", *Programiz.com*. [En línea]. Disponible en: https://www.programiz.com/dsa/selection-sort. [Consultado: 23-nov-2025].