Guía de Actividades Práctico-Experimentales Nro. 005

1. Datos Generales

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:

- 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 y 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

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos).

Inicio

- Presentación breve de objetivos y alcance (qué implementar y cómo verificar).
- Organización individual o en parejas; crear repo Git y rama de trabajo.
- Revisión de firmas y criterios de evaluación (rúbrica).
- Lineamientos de uso responsable de IA.

Desarrollo

- Implementar en este orden: Inserción, Selección, Burbuja (facilita comparar).
- **Estructura del proyecto**
  - Paquete ed.u2.sorting.
  - Clases sugeridas: InsertionSort, SelectionSort, BubbleSort, SortingDemo, SortingUtils.
- **Firmas estándar (Java)**

```
public final class InsertionSort {
        public static void sort(int[] a) { /* ... */ }
}
public final class SelectionSort {
        public static void sort(int[] a) { /* ... */ }
}
public final class BubbleSort {
        public static void sort(int[] a) { /* ... */ }
}
```

  - Todas **in-place** (modifican el arreglo recibido).
  - Agregar sobrecarga opcional sort(int[] a, boolean trace) para imprimir trazas.
- **Inserción (recomendado empezar aquí)**
  - Bucle externo i=1..n-1; insertar a[i] en 0..i-1.
  - **Trazas**: imprimir posiciones movidas y arreglo resultante por iteración.
- **Selección**
  - Para cada i, hallar **mínimo** en i..n-1 y cambiar por a[i].
  - Contabilizar **swaps** (pocos intercambios).
- **Burbuja**
  - Pasadas sucesivas; intercambiar adyacentes “mal ordenados”.
  - Optimización: corte temprano si no hubo **swaps** en una pasada.
- **Casos bordes**
  - Arreglo vacío, tamaño 1, todos iguales, ya ordenado, orden inverso, con duplicados.
- **Trazas y validación**
  - Ejecutar SortingDemo con datasets de la sección 10.
  - Guardar capturas/salidas en TXT para subir al EVA.

Cierre

- Comparación cualitativa: cuándo usar cada algoritmo (datos pequeños, casi ordenados, número de swaps, legibilidad de traza).
- Revisión de evidencias y dudas frecuentes.
- Recordatorio de entregables y modo de verificación automática en EVA.

6. Resultados esperados:

- ZIP con src/ (clases de sort y SortingDemo), y README.md (1 pág.: decisiones, casos borde, cómo ejecutar).
- Evidencias:
- Trazas/outputs (.txt o capturas) de los datasets requeridos.
- Tabla breve comparando recuentos de movimientos/intercambios observados (cualitativo).

7. Preguntas de Control:

Datasets y salidas esperadas:

- A = [8, 3, 6, 3, 9]
- B = [5, 4, 3, 2, 1] (inverso)
- C = [1, 2, 3, 4, 5] (ya ordenado)
- D = [2, 2, 2, 2] (duplicados)
- E = [9, 1, 8, 2]

Orden final esperado (para los tres algoritmos)

- A → [3, 3, 6, 8, 9]
- B → [1, 2, 3, 4, 5]
- C → [1, 2, 3, 4, 5]
- D → [2, 2, 2, 2]
- E → [1, 2, 8, 9]

Preguntas de control:

- ¿Por qué Inserción es preferible con datos casi ordenados?
- ¿Qué propiedad hace que Selección use pocos swaps? ¿Qué compromisos tiene?
- ¿Cómo implementarías el corte temprano en Burbuja y qué caso reduce significativamente?
- ¿Cuál(es) de los tres puede(n) ser estable y en qué condiciones?
- Menciona dos casos borde que deben probarse siempre.

8. Evaluación

Criterio	4 – Excelente	3 – Bueno	2 – Básico	1 – Insuficiente	Pts
Implementación InsertionSort	Correcta; traza clara; maneja casos borde	Detalles menores	Parcial	No funcional	3.0
Implementación SelectionSort	Correcta; contabiliza swaps	Detalles menores	Parcial	No funcional	2.5
Implementación BubbleSort	Correcta; corte temprano implementado	Detalles menores	Parcial	No funcional	2.5
Evidencias (README + salidas)	Completas, reproducibles	Aceptables	Escasas	Inexistentes	1.0
Calidad de código	Nombres/estructura/comentarios adecuados	Aceptable	Pobre	Deficiente	1.0

9. Bibliografía

[1] OpenDSA Project, "Sorting and Searching Modules," Virginia Tech, 2021–2024 (REA con visualizaciones y ejercicios).

[2] P. W. Bible and L. Moser, An Open Guide to Data Structures and Algorithms. PALNI Open Press, 2023.

[3] Oracle, "Java SE 17–21 Documentation: Arrays, Collections, and I/O (java.nio.file)." 2021–2025.

[4] T. Roughgarden, Algorithms Illuminated (Part 1), Soundlikeyourself Publishing, 2019–2020 (con recursos online actualizados).

10. Elaboración y Aprobación

Elaborado por	Andrés R Navas Castellanos **Docente**	Firmado electrónicamente por: ANDRES ROBERTO NAVAS CASTELLANOS Validar únicamente con FirmaEC
Revisado por **Solo si es realizado en laboratorios**	Luis Sinche **Técnico Docente**	No Aplica
Aprobado por	Edison L Coronel Romero **Director de Carrera**	Firmado electrónicamente por: EDISON LEONARDO CORONEL ROMERO Validar únicamente con FirmaEC