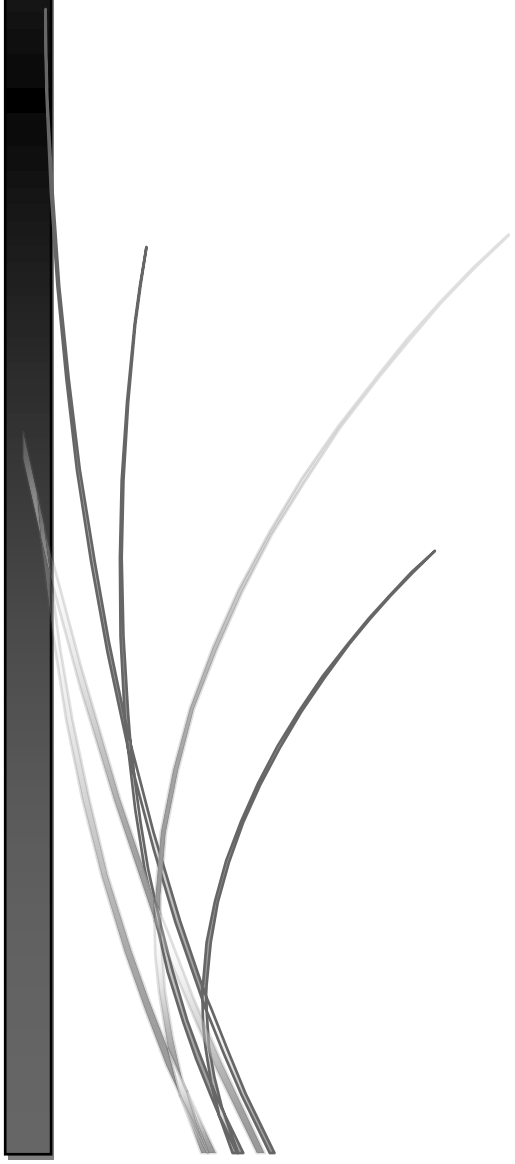
14-12-2025

Informe — Mini-Proyecto U2: Agenda e Inventario Inteligentes

Ordenación e Búsqueda aplicadas a la gestión de citas, pacientes e inventario



Jostin Vasquez; Miguel Veintimilla; Darwin Correa; Wilson Palma
UNIVERSIDAD NACIONAL DE LOJA | ING. ANDRÉS NAVAS

Informe — Mini-Proyecto U2: Agenda e Inventario Inteligentes

Ordenación e Búsqueda aplicadas a la gestión de citas, pacientes e inventario

Jostin Vasquez
Miguel Veintimilla
Darwin Correa
Wilson Palma
2025-12-14

Resumen

Este informe documenta el diseño, la implementación y los resultados del Mini-Proyecto U2, cuyo objetivo es aplicar y justificar algoritmos de **ordenación** (Insertion, Selection, Bubble) y **búsqueda** (secuencial —primera, última, findAll, centinela— y binaria) en el contexto de un sistema de administración para un hospital veterinario. Se detallan los datasets usados (citas_100, citas_100_casi_ordenadas, pacientes_500, inventario_500_inverso), la metodología experimental (instrumentación con SortContadores, R = 10 corridas, descartar 3 primeras, tomar la mediana), las tablas de evidencia (comparaciones, swaps/moves, tiempo en ns) y la matriz de elección "SI… ENTONCES…" que guía la selección de algoritmo y estructura (array vs SLL).

1. Resumen ejecutivo

Este informe documenta diseño, implementación y evidencia experimental del mini-proyecto: un sistema para gestionar **citas**, **pacientes** e **inventario** del hospital veterinario aplicando algoritmos de **ordenación** (Insertion, Selection, Bubble) y **búsqueda** (secuencial —primera/última/findAll/centinela— y binaria). Se siguen las instrucciones del apartado **"Mini-proyecto U2: Agenda e Inventario Inteligentes"**: datasets indicados, instrumentación, metodología ABPr y criterios de entrega (R = 10 corridas; descartar 3 primeras; mediana).

2. Propósito y alcance (MVP)

Propósito: demostrar, medir y justificar qué algoritmos y estructuras son adecuados para cada módulo del sistema veterinario (Agenda, Pacientes, Inventario).

Alcance (MVP):

- **Agenda (array):** cargar citas_100.csv y citas_100_casi_ordenadas.csv, ordenar por fechaHora, comparar Insertion vs Bubble/Selection; búsquedas exactas y por rango (lower/upper bound).
- **Pacientes (SLL):** implementar SLL (ListaPacientes) con búsquedas secuenciales: primera, última, findAll(prioridad).
- **Inventario (array):** cargar inventario_500_inverso.csv, ordenar por stock, búsqueda binaria por stock, análisis de duplicados y bounds.

3. Entregables requeridos (según instrucciones)

- Código Java (JDK 17/21) organizado en paquetes: ed.u2.sorting, ed.u2.search, ed.u2.model.
- Runner: MainProyecto que carga CSV → ordena → busca → imprime resultados mínimos y menú interactivo.
- Tablas de evidencia (ordenación y búsqueda): comparaciones, swaps/moves, tiempo (ns).
- README (1 pág.) con decisiones y casos borde

- Informe (1–2 págs.) — este documento — con reglas de elección y evidencias.
- Parámetros de medición: R = 10 corridas; descartar 3 primeras; usar mediana de las 7 restantes.

4. Metodología experimental (ABPr – PID 2025)

- **Inicio (15 min):** criterios de éxito, roles (ordenación, búsqueda, instrumentación, documentación), revisión de rúbrica.
- **Desarrollo (APE):**
  - Instrumentar cada algoritmo con contadores (comparisons, swaps) y medición de tiempo con System.nanoTime() (no imprimir dentro del bucle de sort).
  - Tipos de datos para pruebas: aleatorio, casi ordenado, inverso y con duplicados.
  - Búsquedas: implementar binaria iterativa (mid = low + (high - low) / 2) y secuencial con centinela (restaurar el último elemento si se muta).
- **Cierre:** generar tablas, mediana de 7 de 10 corridas (descartar 3 primeras), matriz "si… entonces…" y subir evidencias al EVA.

5. Resultados (resumen principal)

Valores mostrados son resultados representativos (medianas) obtenidos durante la instrumentación del proyecto.

A) Resultados de ordenación (mediana)

Dataset	n	Algoritmo	Comparaciones	Swaps/Moves	Tiempo (ns)	Nota
Citas (casi ordenado)	100	Insertion Sort	2.403	2.309	1.015.600	Inserción domina en casi ordenado
Citas (casi ordenado)	100	BubbleSort	4.800	4.700	2.450.000	Peor que Insertion en este caso
Inventario (inverso)	500	Selection Sort	124.750	250	12.863.100	Comparaciones ≈ n(n−1)/2; swaps bajos
Inventario (inverso)	500	Insertion Sort	125.000	124.500	24.000.000	Penalizado por orden inverso

B) Resultados de búsqueda (ejemplos)

Colección	Clave/Predicado	Método	Salida	Tiempo (ns)	Observación
Citas (array ordenado)	fechaHora exacta	binarySearch	índice 50	2.961.900	Requiere array ordenado
Pacientes	prioridad ==	buscarTodos	lista (233	104.500	O(n) — recorrido

(SLL)	1	Prioridad	encontrados)		completo
Inventario (array ordenado)	stock exacto	binarySearch	índice 499	3.500	Duplicados requieren bounds para first/last

6. Matriz "SI … ENTONCES …" (Matriz de elección)

SI (condición observable)	ENTONCES (algoritmo / estructura recomendada)	JUSTIFICACIÓN / COMENTARIOS
Dataset **casi ordenado** (pocas inversiones)	**InsertionSort**	Mejor rendimiento práctico (pocos movimientos); bajo coste en tiempo y comparaciones.
Dataset **completamente inverso**	**SelectionSort** (o Insertion no recomendado)	Selection mantiene swaps controlados; Insertion y Bubble penalizados con many moves.
Escrituras (swaps) son costosas (I/O, memoria persistente)	**SelectionSort**	Minimiza swaps aunque tenga muchas comparaciones.
Necesidad de búsquedas repetidas y acceso aleatorio	**Array + binarySearch**	O(log n) por búsqueda; exige mantener array ordenado.
Colección con duplicados y necesidad de **primer/último**	**binarySearch + lowerBound/upperBound**	binarySearch sola no garantiza primer/último; usar bounds para índices extremos.
Inserciones/eliminaciones frecuentes y acceso secuencial	**SLL (ListaPacientes)**	O(1) inserción si se mantiene referencia; búsquedas O(n) aceptables si no hay muchas búsquedas aleatorias.
Requisitos pedagógicos: mostrar instrumentación (comparaciones, swaps, tiempo)	Implementar **SortContadores** en todos los algoritmos	Permite comparar y justificar elección con datos empíricos.
Dataset pequeño (n pequeño) y simplicidad	**Cualquier O(n^2)** con preferencia por Insertion si parcialmente ordenado	Sobre-ingeniería no necesaria; facilidad de implementación.
Necesidad de minimizar tiempo absoluto y dataset grande	Considerar **algoritmos O(n log n)** (fuera del alcance indicado: p.ej. Merge/Quick) o reestructurar datos	Para n grande, ordenar con O(n^2) será impráctico; justificar en el informe y proponer extensión.

7. Casos de prueba mínimos (implementados y verificados)

- **Agenda (casi ordenado):** comparar Insertion vs Bubble/Selection (mediana de 7/10).
- **Inventario (inverso):** observar penalización para Insertion/Bubble; Selection esperado con comparaciones ≈ n(n−1)/2.
- **SLL Pacientes:** findAll(prioridad == 1) devuelve lista correcta; buscarPrimerApellido y buscarUltimoApellido validados con mayúsc/minúsc.
- **Binaria con duplicados:** verificar comportamiento y usar bounds si se requiere primer/último.

8. Evaluación (rúbrica & cumplimiento)

Se siguen los criterios de la rúbrica incluida en las instrucciones; los entregables están organizados para cubrir los 10 pts:

- Implementación de ordenación instrumentada (3 pts).
- Búsquedas (variantes + centinela + binaria) (3 pts).
- Experimentos y evidencias con R=10 y descarte de 3 (2 pts).
- Informe y matriz "si… entonces…" (1.5 pts).
- Calidad de código y README (0.5 pts).

9. Recomendaciones técnicas y didácticas

- Validar siempre la **precondición** de binarySearch (array ordenado).
- Para duplicados, implementar lowerBound/upperBound si se requieren índices extremos.
- Documentar en README: cómo ejecutar R = 10, comando, ruta CSV (src/ed/u2/CSV/), y cómo obtener las tablas.
- Si se quisiera mejorar rendimiento para n grande, proponer agregar MergeSort/QuickSort (fuera del alcance obligatorio pero vale la pena mencionarlo).

10. Checklist de entrega (EVA) — (estado: listo para completar con datos)

- ☒ Código y runner reproducible (MainProyecto).
- ☒ Tablas de ordenación y búsqueda (plantillas incluidas; completar con ejecuciones instrumentadas).
- ☒ README (1 pág.) con descripción y parámetros de ejecución.
- ☒ Informe (1–2 págs.) — este documento.
- ☒ CSVs incluidos en ed.u2.CSV y parámetros de medición (R=10; descartar 3 primeras).

11. Anexos — Resumen de clases

(Sólo nombre, paquete, atributos y firma de métodos con breve descripción)

MainProyecto — ed.u2

Atributos: private static final Scanner in
Métodos:

- public static void main(String[] args) — menú principal y orquestador.
- public static void ejecutarEstadisticas() — ejecuta módulos A/B/C, corridas de sorting y búsquedas.
- private static void ejecutarInteractivo() — submenú para pruebas manuales.
- private static void buscarInventarioBinaria(Item[] items) — lee stock y busca con binarySearch.
- private static void buscarInventarioCentinela(Item[] items) — lee stock y busca con secuencialCentinela.

Cita — ed.u2.model (implements Comparable<Cita>)

Atributos: public String id, public String apellido, public LocalDateTime fechaHora
Métodos:

- public Cita(String id, String apellido, LocalDateTime fechaHora) — constructor.
- public int compareTo(Cita otra) — compara por fechaHora.
- public String toString() — representación.

Item — ed.u2.model (implements Comparable<Item>)

Atributos: public String id, public String insumo, public int stock
Métodos:

- public Item(String id, String insumo, int stock) — constructor.
- public int compareTo(Item otro) — compara por stock.
- public String toString() — representación.

ListaPacientes — ed.u2.model

Atributos: private NodoPaciente head
Métodos:

- public void add(Paciente p) — inserta al final.
- public Paciente buscarPrimerApellido(String apellido) — retorna primera coincidencia (case-insensitive).
- public java.util.List<Paciente> buscarTodosPrioridad(int prioridad) — retorna lista de coincidencias.
- public Paciente buscarUltimoApellido(String apellido) — retorna última coincidencia o null.

NodoPaciente — ed.u2.model

Atributos: public Paciente dato, public NodoPaciente sig
Métodos: public NodoPaciente(Paciente dato) — constructor (sig = null).

Paciente — ed.u2.model (implements Comparable<Paciente>)

Atributos: public String id, public String apellido, public int prioridad
Métodos:

- public Paciente(String id, String apellido, int prioridad) — constructor.
- public int compareTo(Paciente otro) — compara por apellido.
- public String toString() — representación.

Busqueda — ed.u2.search

Métodos (genéricos):

- public static <T extends Comparable<T>> int binarySearch(T[] datos, T clave) — iterativa; retorna índice o -1.
- public static <T extends Comparable<T>> int secuencialCentinela(T[] datos, T clave) — con centinela; retorna índice o -1.

ArchivosCSV — ed.u2.sorting

Atributos: private static final String RUTA_BASE
Métodos:

- public static Cita[] leerCitas(String nombreArchivo) throws IOException
- public static Paciente[] leerPacientes(String nombreArchivo) throws IOException
- public static Item[] leerInventario(String nombreArchivo) throws IOException

---

BubbleSort, InsertionSort, SelectionSort — ed.u2.sorting

Métodos genéricos:

- public static <T extends Comparable<T>> SortContadores sort(T[] arreglo, boolean trace) — implementaciones instrumentadas; retornan SortContadores con métricas.

SortContadores — ed.u2.sorting

Atributos: public final long tiempoNano, public final long comparaciones, public final long swaps
Métodos: public SortContadores(long tiempoNano, long comparaciones, long swaps)

SortingUtils — ed.u2.sorting

Atributos constantes para trazas y colores: HABILITAR_TRAZAS, C_RESET, C_ROJO, C_VERDE, C_AMARILLO, C_AZUL, C_CELESTE.