Informe Mini Proyecto Unidad 2

Asignatura: Estructura de Datos
Tema: Agenda e Inventario Inteligentes
Ciclo: Tercero
Integrantes:

- Bryan Troya
- Ismael Gonzalez
- Juan Calopino

Introducción

El presente informe documenta el desarrollo de un mini-proyecto académico cuyo objetivo es analizar y comparar algoritmos de ordenación y búsqueda, aplicados a estructuras de datos fundamentales como arreglos y listas simplemente enlazadas (SLL).
El sistema simula una agenda de citas, un inventario de insumos y un registro de pacientes, permitiendo evaluar el comportamiento de distintos algoritmos bajo escenarios controlados de datos (casi ordenados, inversos y con duplicados).

Objetivo general

Analizar el desempeño de algoritmos clásicos de ordenación y búsqueda mediante métricas de tiempo y operaciones básicas.

Objetivos específicos

- Implementar Bubble Sort, Selection Sort e Insertion Sort.
- Medir comparaciones, intercambios (swaps) y tiempo de ejecución.
- Aplicar búsqueda secuencial y búsqueda binaria iterativa.
- Justificar la elección de algoritmos según el estado de los datos.

Metodología

Estructuras de datos utilizadas

- Arreglos: Citas e Inventario.
- Lista simplemente enlazada (SLL): Pacientes.

Algoritmos de ordenación

- Bubble Sort
- Selection Sort
- Insertion Sort

Algoritmos de búsqueda

- Búsqueda secuencial (primera coincidencia, última coincidencia, findAll).
- Búsqueda secuencial con centinela (arreglos).
- Búsqueda binaria iterativa.
- Lower Bound para rangos y duplicados.

Instrumentación y medición

Para cada algoritmo de ordenación se realizaron 10 corridas, descartando las 3 más lentas. Sobre las 7 restantes se calculó la mediana de:

- Tiempo de ejecución (System.nanoTime)
- Número de comparaciones
- Número de intercambios (swaps)

Casos de prueba

Agenda (datos casi ordenados)

- Se comparó Insertion Sort frente a Bubble y Selection.
- Resultado esperado: Insertion Sort presenta menor tiempo y swaps.

Inventario (datos en orden inverso)

- Se evaluó el impacto en los tres algoritmos.
- Resultado esperado: Selection Sort mantiene comparaciones ≈ n(n−1)/2.

Pacientes (SLL)

- Búsqueda del primer y último paciente por apellido.
- Búsqueda de todos los pacientes con prioridad == 1.

Búsqueda binaria con duplicados

- Se evidenció que binarySearch no garantiza el índice exacto.
- Se implementó Lower Bound cuando se requiere un rango.

Resultados

Tabla A – Ordenación (mediana de 10 corridas)

```
=== TABLA A: ORDENACIÓN ===

------------------------------------------------------------
A) Resultados de ordenacion (Mediana de 10 Corridas)
------------------------------------------------------------
| Dataset (n)          | Algoritmo       | Comparaciones   | Swaps           | Tiempo (ns)     |
------------------------------------------------------------
| Citas (100, Casi Ordenado) | BubbleSort      | 4797            | 341             | 240600          |
| Citas (100, Casi Ordenado) | SelectionSort   | 4950            | 5               | 219400          |
| Citas (100, Casi Ordenado) | InsertionSort   | 440             | 341             | 28800           |
| Inventario (500, Inverso) | BubbleSort      | 124750          | 124750          | 1087800         |
| Inventario (500, Inverso) | SelectionSort   | 124750          | 250             | 571600          |
| Inventario (500, Inverso) | InsertionSort   | 124750          | 124750          | 957700          |
```

- En datos casi ordenados, Insertion Sort fue el más eficiente.
- En datos inversos, Selection Sort mantuvo comparaciones constantes.
- Bubble Sort mostró mayor número de swaps en todos los escenarios.

Tabla B – Búsqueda

```
=== TABLA B: BÚSQUEDA ===

--- Búsqueda en SLL y Arreglos ---
-------------------------------------------------------------------------------------------------------------------------------------------
| Colección          | Clave/Predicado           | Método            | Salida                                                    | Correcto |
-------------------------------------------------------------------------------------------------------------------------------------------
| SLL Pacientes      | Apellido 'Torres' (1er)   | Secuencial        | Paciente{PAC-0029 | Apellido: Torres | Prio: 3}    | Si
| SLL Pacientes      | Apellido 'Torres' (Últ)   | Secuencial        | Paciente{PAC-0467 | Apellido: Torres | Prio: 3}    | Si
| SLL Pacientes      | Prioridad == 1            | Secuencial (FindAll) | Total: 233. Ej: Zambrano                                | Si
| Arreglo Inventario | Stock: 50                 | Centinela         | Índice 49 (Insumo: 50)                                    | Si
| Arreglo Citas      | Fecha exacta              | Binaria           | Índice 48 (Cita: 2025-03-14T11:30)                        | Si
| Arreglo Inventario | Stock >= 200 (Lower Bound) | Binaria           | Índice 199 (Stock: 200)                                   | Si
-------------------------------------------------------------------------------------------------------------------------------------------
```

- La búsqueda secuencial fue adecuada para SLL.
- La búsqueda binaria redujo el tiempo en arreglos ordenados.
- El uso de Lower Bound permitió manejar duplicados correctamente.

Matriz de elecciones (Si… entonces …)

Escenario	Entonces se utiliza	Justificación
Datos casi ordenados	Insertion Sort	Aprovecha el orden previo y reduce comparaciones y swaps (mejor caso ≈ O(n)).
Datos inversos	Selection Sort	Mantiene comparaciones constantes ≈ n(n−1)/2 y pocos swaps.
Lista enlazada	Búsqueda secuencial	No hay acceso aleatorio.
Se buscan todas las coincidencias	Secuencial – findAll	Permite recorrer toda la estructura.
Arreglo ordenado	Búsqueda binaria	Reduce el tiempo a O(log n).
Duplicados	Lower Bound	La binaria estándar no garantiza el índice exacto por ello se implementa el lower bound.

Conclusiones

- Los resultados obtenidos confirman que el comportamiento de los algoritmos de ordenación depende directamente del estado inicial de los datos. En conjuntos casi ordenados, el algoritmo de Inserción presenta el mejor desempeño al reducir significativamente comparaciones y movimientos, acercándose a su mejor caso $O(n)$ O(n). En contraste, en datos ordenados de forma inversa, Burbuja e Inserción muestran una penalización cuadrática, mientras que Selección mantiene un número constante de comparaciones cercano a $n(n-1)/2$ n(n−1)/2, lo que valida su comportamiento teórico.
- En los experimentos de búsqueda, la búsqueda secuencial sobre listas simplemente enlazadas permitió obtener correctamente la primera y última coincidencia, así como múltiples resultados mediante findAll, demostrando su utilidad cuando no existe orden previo. Por su parte, la búsqueda binaria en arreglos ordenados mostró eficiencia superior, aunque en duplicados, el índice retornado no está garantizado, lo que justificó la implementación de variantes como lowerBound.
- Finalmente, la instrumentación mediante contadores de comparaciones, swaps y medición con System.nanoTime(), junto con el uso de la mediana de las mejores corridas, permitió obtener resultados estables y confiables.