Guía de Actividades Práctico Experimentales Nro. 006

1. Datos Generales

Nombre del estudiante(s)	-Alex Sigcho -Ivett Zaragocin
Asignatura	Estructura de datos
Ciclo	3er “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 20 de noviembre Viernes 21 de noviembre

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 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 ofimáticas (Google Docs/LibreOffice/Word).

5. Procedimiento / Metodología

Para asegurar que los resultados sean válidos se siguió un método el cual se dividió en cuatro etapas:

A. Creación Controlada de Datos

Se elaboraron scripts que generaron cuatro conjuntos de datos sintéticos con características determinadas, utilizando un valor aleatorio fijo. Esto asegura que los resultados sean reproducibles

Citas (n=100): Distribución uniforme y aleatoria (fechas y horas).

Citas Casi Ordenadas (n=100): Conjunto de datos ordenado con un 5% de alteración (5 pares intercambiados). Diseñado para evaluar la adaptabilidad.

Pacientes (n=500): Alta cantidad de duplicados (pocos apellidos únicos) para analizar la estabilidad

Inventario (n=500): Orden estrictamente opuesto (descendente). Representa el peor escenario teórico.

B. Instrumentación de Algoritmos

Se llevaron a cabo implementaciones de los algoritmos Bubble Sort (con optimización de corte anticipado), Insertion Sort y Selection Sort, utilizando Genéricos en Java. Para recoger datos, se utilizó una técnica de medición no intrusiva:

Se inyectó un objeto llamado Metrics en cada uno de los algoritmos.

```java
public class Burbuja {  1 usage  ivettzaragocin *
    public static <T extends Comparable<T>> void sort(T[] a, Metrica metrica) {
        long start = System.nanoTime();
        int n = a.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                metrica.comparisons++;
                if (a[j].compareTo(a[j + 1]) > 0) {
                    metrica.swaps++;
                    T temp = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = temp;
                    swapped = true;
                }
            }           if (!swapped) break;
        }
        long end = System.nanoTime();
        metrica.timeNs = end - start;
    }
}
```

C. Diseño Experimental y Aislamiento

Repeticiones (R=10): Cada experimento se realizaba 10 veces seguidas sobre copias limpias de los datos.

Métrica Central: Se reportó la mediana de las 7 ejecuciones restantes (no el promedio) para eliminar picos de latencia (outliers) provocados por procesadores en segundo plano o el recolector de basura

D. Entorno de Ejecución

Las pruebas fueron realizadas midiendo el tiempo de en nanosegundos validando el rendimiento de los algoritmos con diferentes tipos de datos (fechas, cadenas de texto y números enteros).

1. ¿Cuándo conviene usar cada algoritmo?

Debate sobre Insertion Sort:

Notamos que en el dataset de *Citas Casi Ordenadas*, InsertionSort fue rápido comparado con los otros dos. Acordamos que Insertion Sort es el más conveniente para datos pre-ordenados.

Debate sobre Selection Sort:

En el dataset de *Inventario Inverso*, aunque Selection Sort tardó bastante tiempo (similar a los otros), nos fijamos en la columna de SWAPS. Mientras Bubble e Insertion hacían miles de movimientos, Selection hizo solo 499

intercambios (N-1). Selection Sort es más conveniente cuando escribir en memoria es costoso.

Debate sobre Bubble Sort:

Vimos que con la optimización de "corte temprano", Bubble Sort se comportó decentemente en datos casi ordenados, pero fue un desastre absoluto en el *Inventario Inverso*. Coincidimos en que Bubble Sort es principalmente pedagógico.

2. ¿Qué sesgos introdujo la medición?

Durante la experimentación, identificamos factores externos que influyen en las mediciones, como el ruido del sistema operativo, donde procesos en segundo plano generan latencia aleatoria, y el sesgo de calentamiento de la JVM, que hace las primeras ejecuciones más lentas. Además, existe un leve sesgo de instrumentación, ya que el propio código encargado de contar las comparaciones añade una sobrecarga computacional mínima pero constante al tiempo total del algoritmo

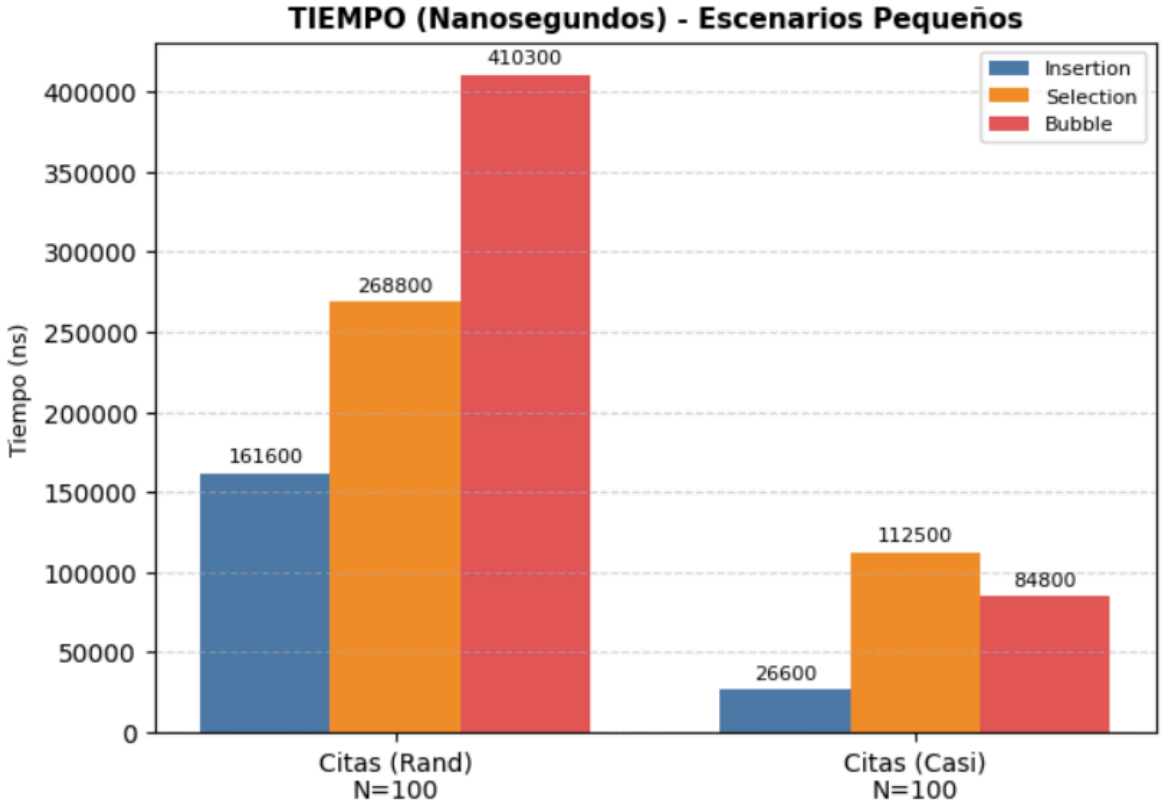
6. Resultados

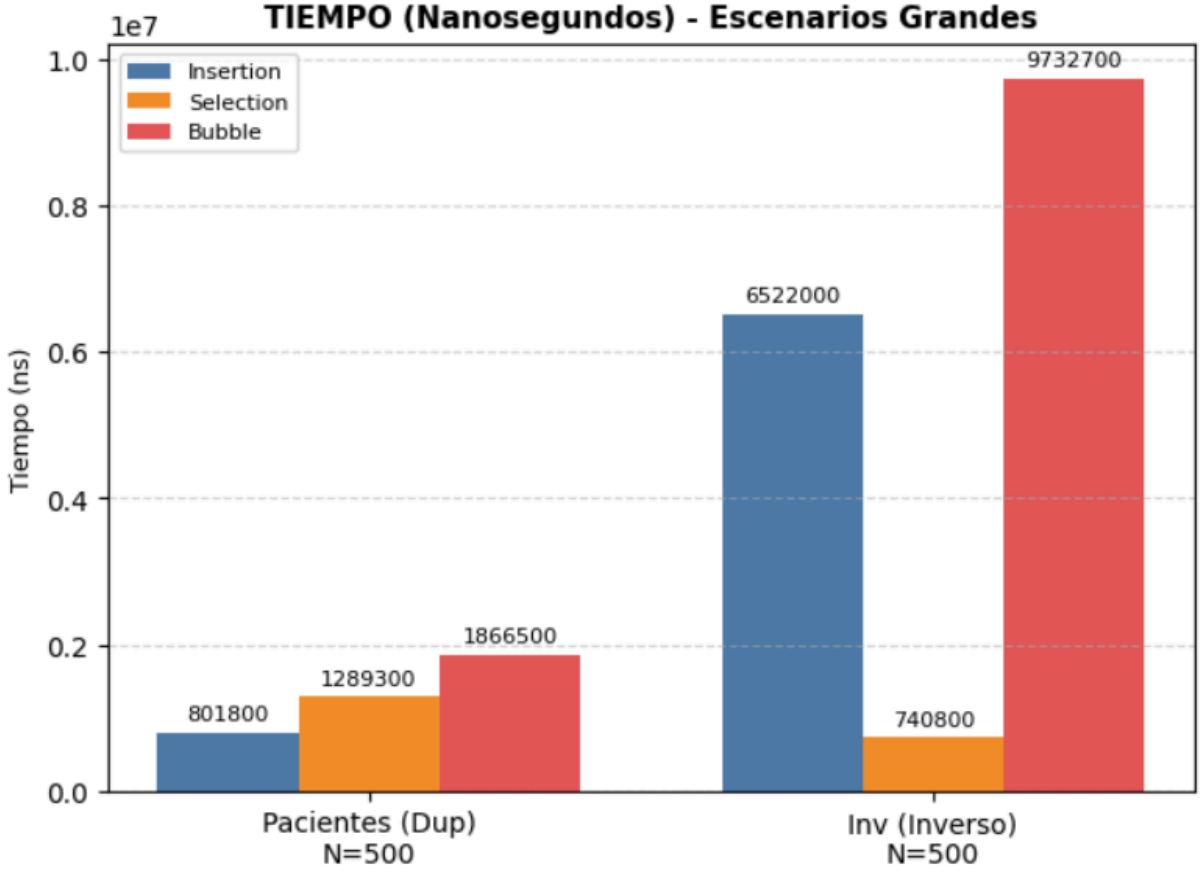
- **Tabla por dataset: `n, tipo (aleat/casi-ord/dup/inverso), algoritmo, comparisons, swaps, tiempo_mediana(ns)`.**

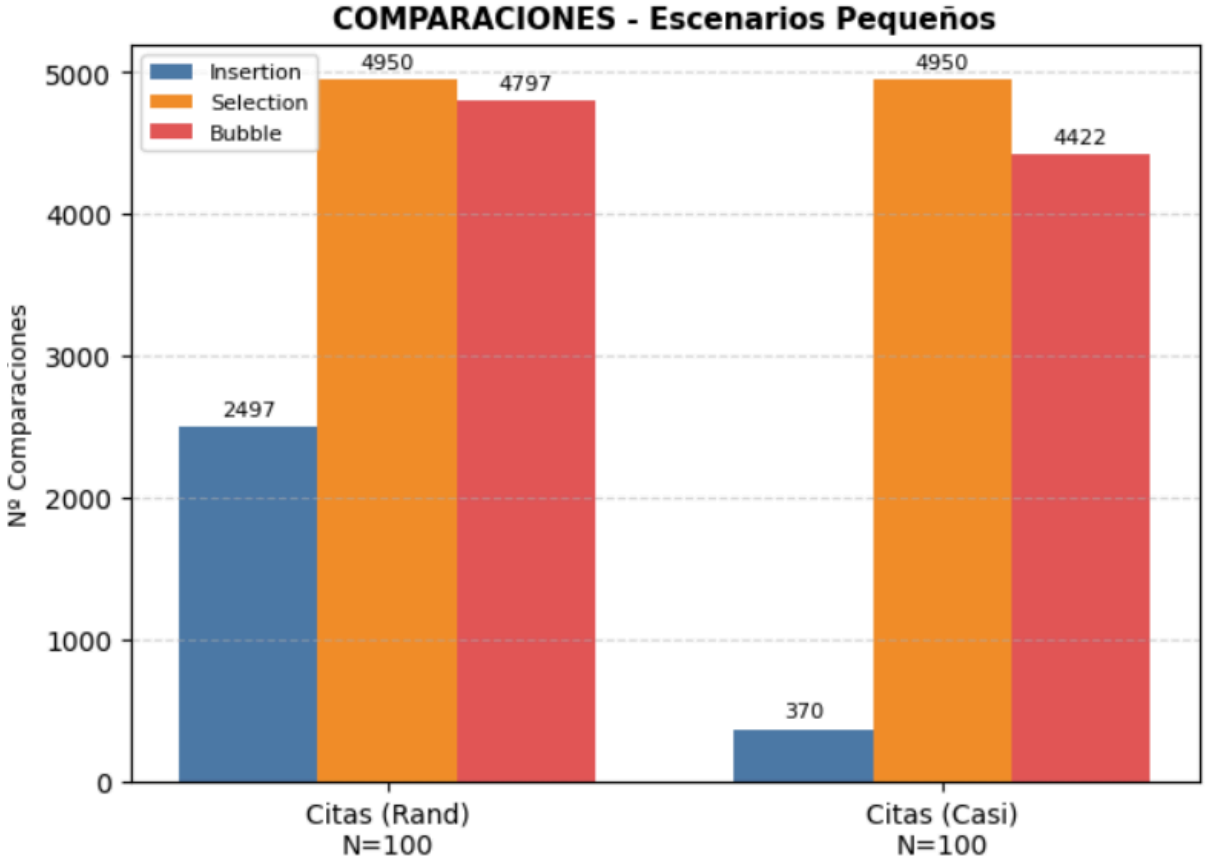
N	Tipo (Dataset)	Algoritmo	Comparisons	Swaps	Tiempo Mediana (ns)
100	Aleatorio (Citas)	Insertion	2,497	2,401	161,600
100	Aleatorio (Citas)	Selection	4,950	95	268,800
100	Aleatorio (Citas)	Bubble	4,797	2,401	410,300
100	Casi Ordenado (Citas)	Insertion	370	271	26,600
100	Casi Ordenado (Citas)	Selection	4,950	5	112,500
100	Casi Ordenado (Citas)	Bubble	4,422	271	84,800
500	Duplicados (Pacientes)	Insertion	51,805	51,307	801,800
500	Duplicados (Pacientes)	Selection	124,750	397	1,289,300

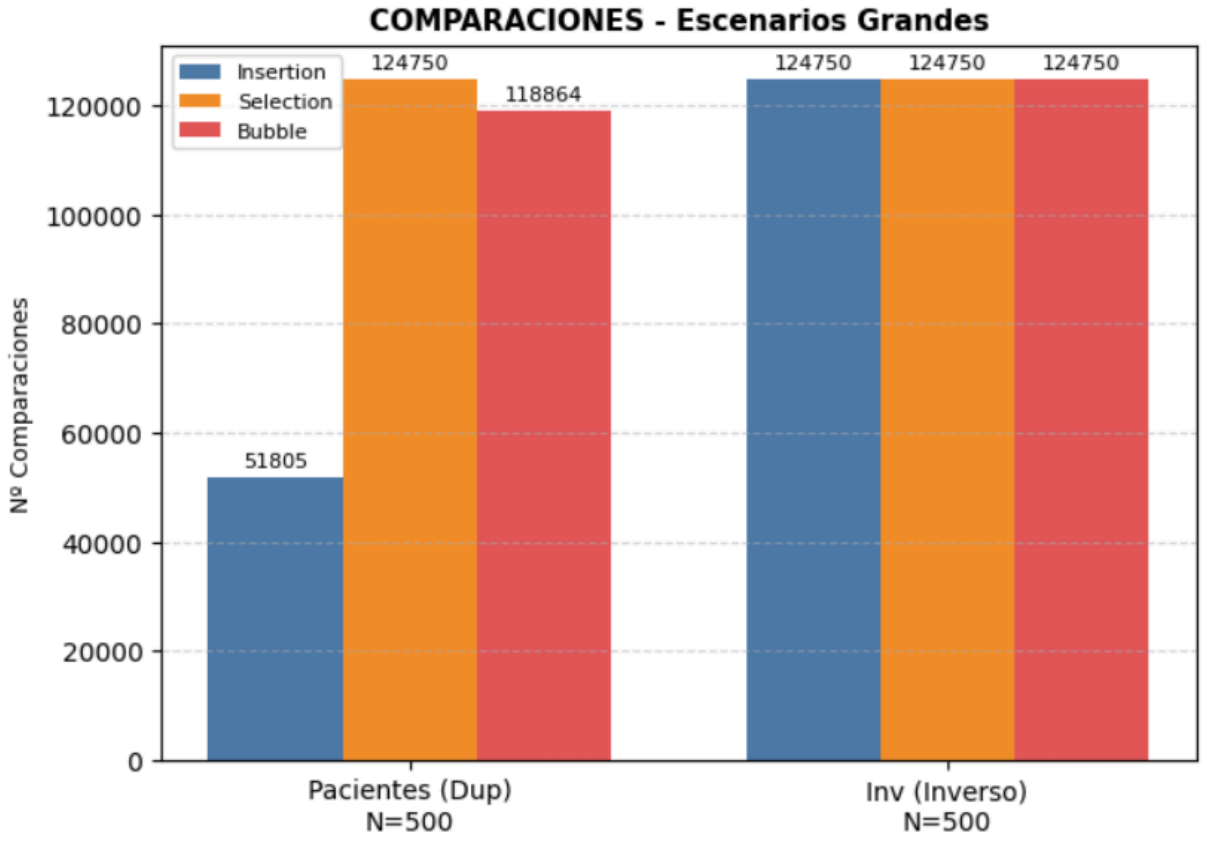
500	Duplicados (Pacientes)	Bubble	118,864	51,307	1,866,500
500	Inverso (Inv)	Insertion	124,750	124,750	6,522,000
500	Inverso (Inv)	Selection	124,750	250	740,800
500	Inverso (Inv)	Bubble	124,750	124,750	9,732,700

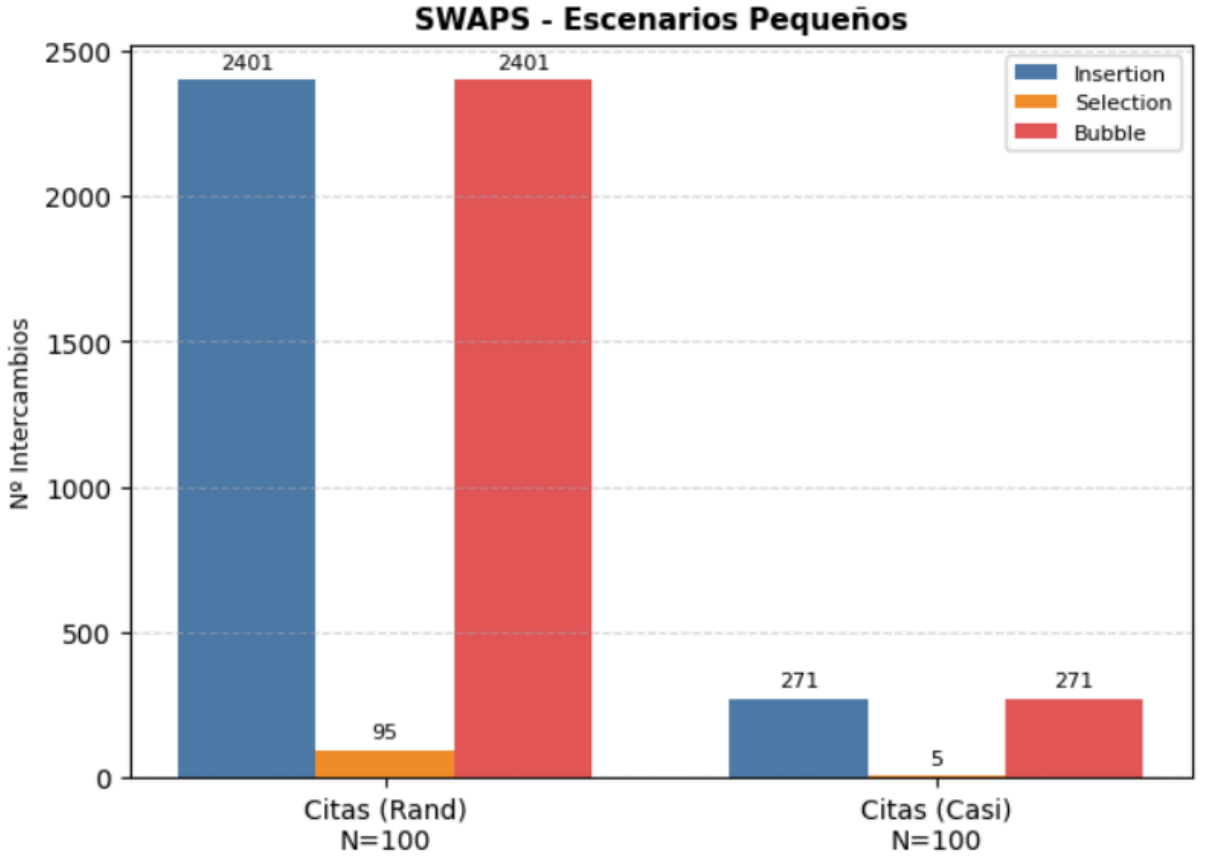
- **Gráficos (opcional): barras o líneas para tiempo y comparaciones.**

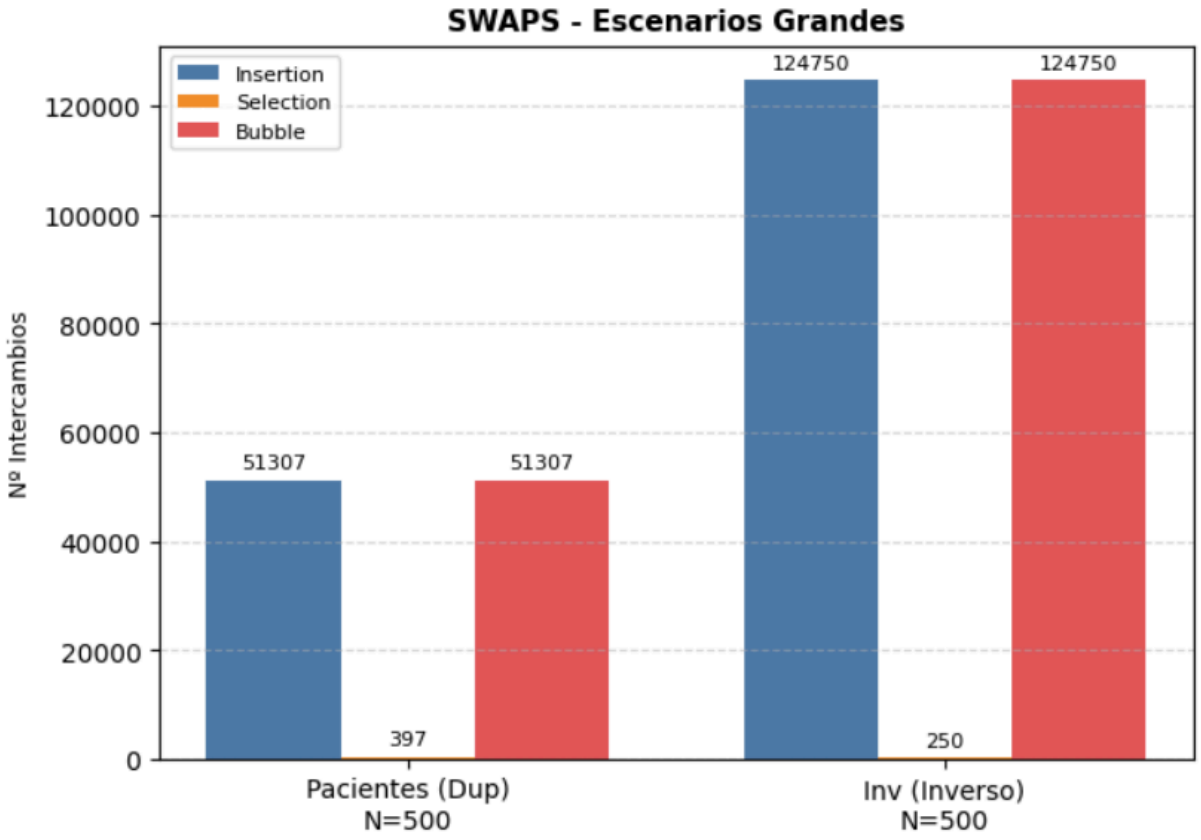



TIEMPO (Nanosegundos) - Escenarios Grandes
1e7
Tiempo (ns)
Insertion
Selection
Bubble
801800
1289300
1866500
6522000
740800
9732700
Pacientes (Dup)
N=500
Inv (Inverso)
N=500


COMPARACIONES - Escenarios Pequeños
Nº Comparaciones
Insertion
Selection
Bubble
2497
4950
4797
370
4950
4422
Citas (Rand)
N=100
Citas (Casi)
N=100


COMPARACIONES - Escenarios Grandes
Insertion
Selection
Bubble
Nº Comparaciones
0
20000
40000
60000
80000
100000
120000
51805
124750
118864
124750
124750
124750
Pacientes (Dup)
N=500
Inv (Inverso)
N=500


SWAPS - Escenarios Pequeños
Insertion
Selection
Bubble
Nº Intercambios
0
500
1000
1500
2000
2500
2401
95
2401
271
5
271
Citas (Rand)
N=100
Citas (Casi)
N=100



- **Matriz de recomendación (texto/tabla): "si casi ordenado y n≤500 → Inserción", "si minimizar swaps → Selección", etc.**

Escenario	Algoritmo	¿Por qué?
Datos están casi ordenados	**Insertion Sort**	Fue el más rápido ya que lo hizo en 26k ns además de sólo 370 comparaciones cuando los demás hicieron 4000 comparaciones.
Menos intercambios (Swaps)	**Selection Sort**	Aunque es el más lento hace ya que en random hace muy pocos intercambios (95 vs 2400 de los otros ordenadores)
Datos pequeños (N=100) aleatorios	**Insertion Sort**	Este algoritmo tuvo mejor rendimiento en lo que respecta a tiempo (161k ns) que los algoritmos de Selection y Bubble.
Peor caso cuando tienen orden inverso	**Ninguno (o Selection)**	Los 3 algoritmos terminaron en el mismo número de comparaciones pero Selection fue ligeramente más rápido en tiempo que Bubble y menos intercambios

- **Capturas/Logs de ejecución (sin trazas durante medición).**

```
============ TALLER 6: COMPARACIÓN DE ORDENACIÓN =============
ALGORITMO    DATASET              N     TIEMPO(ns)   COMPS        SWAPS
-------------------------------------------------------------------------------
Insertion    Citas(Rand)          100   161600       2497         2401
Selection    Citas(Rand)          100   268800       4950         95
Bubble       Citas(Rand)          100   410300       4797         2401
-------------------------------------------------------------------------------
Insertion    Citas(Casi)          100   26600        370          271
Selection    Citas(Casi)          100   112500       4950         5
Bubble       Citas(Casi)          100   84800        4422         271
-------------------------------------------------------------------------------
Insertion    Pacientes(Dup)       500   801800       51805        51307
Selection    Pacientes(Dup)       500   1289300      124750       397
Bubble       Pacientes(Dup)       500   1866500      118864       51307
-------------------------------------------------------------------------------
Insertion    Inv(Inverso)         500   6522000      124750       124750
Selection    Inv(Inverso)         500   740800       124750       250
Bubble       Inv(Inverso)         500   9732700      124750       124750
-------------------------------------------------------------------------------
```

- **Código con instrumentación y scripts de generación de datasets (si aplica)**

  Link GitHub: https://github.com/Alex-Francis-07/Taller_6_ED

7. Preguntas de Control:

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

  Cuando se imprime en consola con System.out.println esta es muy lenta comparada a las operaciones que realiza el procesador y cuando el algoritmo imprime un mensaje el programa debe de detener el cálculo matemático, enviar la cadena al buffer del sistema operativo y esperar a que la terminar muestre el mensaje esto hace que exista una latencia y si se mide el tiempo con trazas activas en realidad se medirá la velocidad de la consola no lo que de verdad se quiere medir que es la eficiencia del algoritmo de ordenación.

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

  El algoritmo de selección es un algoritmo no adaptativo ya que su lógica consiste en dos bucles anidados que son fijos y el bucle externo va a recorre desde i = 0 hasta n - 1 y el bucle interno siempre recorre

desde j = i + 1 hasta el final de la lista para buscar un dato que sea el mínimo y esto lo hace sin importar si la lista ya está ordenada.

- **¿Por qué Inserción es competitivo en datos casi ordenados?**
  Porque cuando compara los datos hacia atrás y ve que el dato ya se encuentra en orden el algoritmo se detiene inmediatamente y no sigue comparando los datos ya ordenados y no pierde tiempo.
- **¿Qué papel juegan los duplicados en la estabilidad del resultado?**
  Los duplicados evitan que se hagan intercambios (swaps) si el algoritmo encuentra otro dato igual y de esta manera evita gastar tiempo al momento de moverlos de lugar.
- **¿Por qué Burbuja con corte temprano mejora en "casi ordenado" pero no en "inverso"?**
  Porque es un lista inversa y el número más pequeño está al final de la lista de datos y el algoritmo de burbuja solo mueve un dato por cada pasada completa haciendo que tenga que hacer todas las pasadas posibles para ordenarlo y no puede terminar el algoritmo de manera temprana.

8. Conclusiones

Gracias a la realización de esta practica se logró obtener las siguientes conclusiones:

- Insertion Sort es el algoritmo más fuerte para situaciones generales y de la vida real. Su habilidad para identificar el orden ya presente permite que logre una complejidad cercana a O(n) en el conjunto de datos Citas Casi Ordenadas, superando con creces a Selection Sort, que no reconoce el orden y siempre realiza O(n^2) comparaciones.
- Aunque los tres algoritmos teóricamente se consideran como O(n^2), se demostró que las operaciones no tienen el mismo costo. Selection Sort, a pesar de ser lento se mostró como el más eficiente en cuanto a escritura en memoria, haciendo un máximo de N intercambios.
- Bubble Sort confirmó su debilidad inherente en el conjunto de datos Inventario Inverso.

9. Recomendaciones

- Realizar más tipos de pruebas para verificar que realmente los algoritmos funcionen de la manera discutida

10. 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`), and benchmarking notes,” 2021–2025.

[4] OpenJDK, “JMH – Java Microbenchmark Harness: Samples and Guidance,” 2020–2025 (guía práctica de mediciones reproducibles).