Reporte Técnico de Actividades Práctico-Experimentales Nro. 007

1. Datos de Identificación del Estudiante y la Práctica

Nombre del estudiante(s)	- Mark Gonzales - Alejandro Padilla - Gyna Yupanqui - Steven Jumbo
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	Búsqueda en Java: Secuencial y Binaria
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 27 de noviembre
Horario	07h30 – 10h30
Lugar	Aula
Tiempo planificado en el Sílabo	3 horas

2. Objetivo(s) de la Práctica

- Implementar correctamente las variantes canónicas de búsqueda secuencial y búsqueda binaria en Java.
- Validar con casos borde, y justificar cuándo aplicar cada método según la estructura de datos (arreglo vs SLL).

3. Materiales, Reactivos

- Datasets.

4. Equipos y Herramientas

- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión “Extension Pack for Java”) o IntelliJ IDEACommunity.
- 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. Primera ocurrencia (array y SLL)
  - Arrays: int indexOfFirst(int[] a, int key) → retornar al primer match.
  - SLL: Node findFirst(Node head, int key) → retornar nodo al primer match.
  - Casos borde: vacío, uno solo, duplicados (en índice 0, medio, final).
- Paso 2. Última ocurrencia (array y SLL)
  - Arrays: una pasada guardando last actualizado; o de atrás hacia adelante.
  - SLL: una pasada guardando Node last.
  - Casos: sin apariciones, todas las posiciones coinciden.
- Paso 3. . findAll por predicado (array y SLL)
  - Arrays: List<Integer> findAll(int[] a, IntPredicate p)
  - SLL: List<Node> findAll(Node head, Predicate<Node> p)
  - Predicados sugeridos: “par”, “==key”, “< umbral”.
  - Salida: lista de índices (array) / nodos (SLL).
- Paso 4. Secuencial con centinela (solo arrays)
  - Técnica: guardar el último elemento, escribir key al final, bucle sin chequeo de límites, restaurar último, decidir si fue hallazgo real o por centinela.
  - Comparar comparaciones realizadas vs. variante clásica.
- Paso 5. Búsqueda binaria (arrays ordenados)
  - int binarySearch(int[] a, int key) (iterativa).
  - Cuidados: mid = low + (high - low) / 2, precondición de arreglo ordenado.
  - Opcional (plus): lowerBound/upperBound para primera/última con duplicados.
- Paso 6. Pruebas y verificación
  - Ejecutar SearchDemo con:
  - Arrays: A, B, C, D; claves: 7, 5, 2, 42 (no está).

- SLL: 3→1→3→2, claves: 3 (primera/última) y predicado val<3.
- Registrar índices/nodos esperados y observados.
- Evidencias: tabla con entradas, método y salida.

Cierre

- Discusión guiada: Cuando conviene secuencial vs binaria; centinela en "no encontrado."
- Completar README e informe con evidencias y decisiones.

6. Resultados

Archivo con Inventario Inverso

```
=== LABORATORIO 7: ALGORITMOS DE BÚSQUEDA ===
=== Integrantes del Grupo: Mark Gonzáles, Steven Jumbo, Alejandro Padilla, Gyna Yupanqui ===


================================================
 PRUEBA DE ARCHIVO: Inventario Inverso
 Archivo: src/main/java/data/inventario_500_inverso.csv
 Buscando clave: 45.0
================================================
 -> Datos cargados. Total elementos: 500
1. Secuencial (First): Encontrado [45.0] en índice 455
   Tiempo: 1277600 ns
2. Secuencial (Last) : Encontrado [45.0] en índice 455
   Tiempo: 8200 ns
3. Centinela         : Encontrado [45.0] en índice 455
   Tiempo: 20700 ns
   [Sistema] Ordenando datos (BubbleSort) para Búsqueda Binaria...
4. Binaria (con Sort): Encontrado [45.0] en índice 44
   Tiempo: 8506900 ns (Incluye ordenamiento)
5. SLL FindAll: Encontrados 1 nodos.
   Tiempo: 1709800 ns
```

Archivo con Números Positivos

```
================================================
 PRUEBA DE ARCHIVO: Números Positivos
 Archivo: src/main/java/data/NumerosPositivos.csv
 Buscando clave: 999.0
================================================
 -> Datos cargados. Total elementos: 98
1. Secuencial (First): Encontrado [999.0] en índice 79
   Tiempo: 7300 ns
2. Secuencial (Last) : Encontrado [999.0] en índice 79
   Tiempo: 2200 ns
3. Centinela         : Encontrado [999.0] en índice 79
   Tiempo: 4800 ns
   [Sistema] Ordenando datos (BubbleSort) para Búsqueda Binaria...
4. Binaria (con Sort): Encontrado [999.0] en índice 97
   Tiempo: 485700 ns (Incluye ordenamiento)
5. SLL FindAll: Encontrados 1 nodos.
   Tiempo: 31000 ns
```

Archivo con Números Duplicados

```
================================================
 PRUEBA DE ARCHIVO: Números Duplicados
 Archivo: src/main/java/data/NumerosDuplicados.csv
 Buscando clave: 12.0
================================================
 -> Datos cargados. Total elementos: 95
1. Secuencial (First): Encontrado [12.0] en índice 0
   Tiempo: 2000 ns
2. Secuencial (Last) : Encontrado [12.0] en índice 64
   Tiempo: 3200 ns
3. Centinela         : Encontrado [12.0] en índice 0
   Tiempo: 1400 ns
   [Sistema] Ordenando datos (BubbleSort) para Búsqueda Binaria...
4. Binaria (con Sort): Encontrado [12.0] en índice 14
   Tiempo: 225300 ns (Incluye ordenamiento)
5. SLL FindAll: Encontrados 4 nodos.
   Tiempo: 42800 ns
```

Archivos con Positivos y Negativos

```
================================================
 PRUEBA DE ARCHIVO: Positivos y Negativos
 Archivo: src/main/java/data/NumerosPositivosNegativos.csv
 Buscando clave: -5.0
================================================
 -> Datos cargados. Total elementos: 99
1. Secuencial (First): Encontrado [-5.0] en índice 1
   Tiempo: 1700 ns
2. Secuencial (Last) : Encontrado [-5.0] en índice 1
   Tiempo: 6300 ns
3. Centinela         : Encontrado [-5.0] en índice 1
   Tiempo: 1500 ns
   [Sistema] Ordenando datos (BubbleSort) para Búsqueda Binaria...
4. Binaria (con Sort): Encontrado [-5.0] en índice 34
   Tiempo: 512500 ns (Incluye ordenamiento)
5. SLL FindAll: Encontrados 1 nodos.
   Tiempo: 16100 ns
```

Casos Borde

```
################################################
       VERIFICACIÓN DE CASOS BORDE (EXTREMOS)
################################################

--- CASO BORDE 1: ARREGLO VACÍO ---
Prueba: Buscar 10.0 en arreglo []
FindFirst: OK (Manejado)
Centinela: OK (Manejado)
Binaria:   OK (Manejado)

--- CASO BORDE 2: UN SOLO ELEMENTO (EXISTE) ---
Prueba: Buscar 50.0 en arreglo [50.0]
[Info] El arreglo tiene un solo elemento (Caso borde).
[Info] El arreglo tiene un solo elemento (Caso borde).
   [Sistema] Ordenando datos (BubbleSort) para Búsqueda Binaria...
FindFirst index: 0 [CORRECTO]
Binaria index:   0 [CORRECTO]

--- CASO BORDE 3: ELEMENTO NO EXISTENTE ---
Centinela buscando 99.0 en [10,20,30]: -1 [CORRECTO]

--- CASO BORDE 4: BÚSQUEDA DE NULL ---
Buscar null: -1 (Debe ser -1 y no crashear) -> OK
[Advertencia] El arreglo está vacío.
[Advertencia] El arreglo está vacío.
[Advertencia] El arreglo está vacío.
```

Enlace al Git

https://github.com/AlejandroTatum/AlgoritmosDeBusqueda.git

7. Preguntas de Control

- **¿Por qué la binaria no es adecuada para SLL aunque esté ordenada?**

  Porque en SLL no se puede acceder a elementos por índice en tiempo constante ya que la búsqueda binaria necesita poder saltar a la mitad de la estructura en tiempo constante, pero en una SLL solo se puede avanzar nodo por nodo, lo que hace que la búsqueda binaria sea ineficiente.

- **En primera ocurrencia, ¿ por qué se retorna en cuanto se encuentra?**

  Porque la estructura ya se encuentra ordenada o el algoritmo garantiza que se recorre de izquierda a derecha, entonces la primera coincidencia encontrada es necesariamente la primera posición válida.

- **¿Qué garantiza la correctitud de la variante centinela?**

  Garantiza que siempre habrá un nodo adicional que simplifica las condiciones de parada, lo que elimina comparaciones extras por límites y asegura que el ciclo va a terminar cuando se encuentre una coincidencia.

- **¿Cómo adaptarías la binaria para duplicados (primera/última)?**

  Se podría adaptar modificando la condición cuando se encuentra un match:

  - **Primera ocurrencia**

    Cuando `mid` coincide, guardar esa posición y continuar buscando a la izquierda (`high = mid - 1`).

  - **Última ocurrencia**

    Cuando `mid` coincide, guardar esa posición y continuar buscando a la derecha (`low = mid + 1`).

- **Propón dos casos borde que hayan detectado errores en tus pruebas.**

  - Elementos buscados primer/último del arreglo.
  - Un arreglo con todos los elementos iguales.

8. Conclusiones

- Una vez terminado el taller se pudo observar que la búsqueda secuencial es flexible y funciona incluso sin orden, mientras que la búsqueda binaria requiere estrictamente un arreglo ordenado para mantener su eficiencia.
- Los casos borde fueron de gran ayuda para validar el comportamiento real de cada variante, logrando detectar errores lógicos.
- La variantes de primera ocurrencia y centinela permitieron optimizar los recorridos del algoritmo.

9. Recomendaciones

- Asegurarse que las precondiciones se cumplan antes de ejecutar una búsqueda binaria, verificando que el arreglo este ordenado para evitar fallos.

- Analizar la estructura de los datos antes de elegir el algoritmo a utilizar.
- Para una mejor visualización siempre añadir casos borde que ya ayudan a detectar errores.

10. Bibliografía / Referencias

[1] "Búsqueda binaria", *Java*, 30-ago-2022. [En línea]. Disponible en: https://codigojava.online/busqueda-binaria-java/. [Consultado: 27-nov-2025].

[2] "Binary search in java", *GeeksforGeeks*, 04-ago-2018. [En línea]. Disponible en: https://www.geeksforgeeks.org/java/binary-search-in-java/. [Consultado: 27-nov-2025].

[3] "Algoritmos de búsqueda", *Github.io*. [En línea]. Disponible en: https://martipatgra.github.io/programacionJava/ud4/6searcharrays/. [Consultado: 27-nov-2025].