Guía de Actividades Práctico Experimentales Nro. 007

1. Datos Generales

Integrantes	Doménica Anahí Rojas Curimilma Victor Fernando Roa Carrión Yandri Alexander Piscocama Jaramillo Jaime Alejandro Landazuri Román
Grupo	F
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 y reactivos:**
   - Datasets.

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 del objetivo y criterios de éxito.
   - Formación de equipos de 4 personas 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: cuándo conviene secuencial vs binaria; centinela en “no encontrado”.
- Completar README e informe con ias y decisiones.

6. Resultados esperados:

Repositorio en Git:

El proyecto se desarrolló utilizando GitHub, manteniendo una estructura ordenada y commits progresivos.

https://github.com/Jaimewww/taller_busqueda_f.git

estudiantes_notas.csv

ARRAY

- colección: 100 registros de notas
- clave/predicado: [82, 85, 88]
- método y salida:

```
Buscando clave: 82
--------------------------------------------------
Primera coincidencia: valor = 82, índice = 9
Última coincidencia: valor = 82, índice = 78
findAll: índices = [ 9 44 78 ]
Secuencial con centinela: encontrado = 82
Búsqueda binaria: índice = 39

Buscando clave: 85
--------------------------------------------------
Primera coincidencia: valor = 85, índice = 0
Última coincidencia: valor = 85, índice = 93
findAll: índices = [ 0 8 15 31 50 58 77 93 ]
Secuencial con centinela: encontrado = 85
Búsqueda binaria: índice = 49

Buscando clave: 88
--------------------------------------------------
Primera coincidencia: valor = 88, índice = 5
Última coincidencia: valor = 88, índice = 95
findAll: índices = [ 5 19 39 59 79 95 ]
Secuencial con centinela: encontrado = 88
Búsqueda binaria: índice = 67
```

- colección: 100 registros de edades
- clave/predicado: [15, 20, 21]
- método y salida:

```
Buscando clave: 15
--------------------------------------------------
Primera coincidencia no encontrada
Última coincidencia no encontrada
findAll vacío
Secuencial con centinela no encontrado
Búsqueda binaria no encontrado

Buscando clave: 20
--------------------------------------------------
Primera coincidencia: valor = 20, índice = 0
Última coincidencia: valor = 20, índice = 96
findAll: índices = [ 0 6 10 14 18 22 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 ]
Secuencial con centinela: encontrado = 20
Búsqueda binaria: índice = 61

Buscando clave: 21
--------------------------------------------------
Primera coincidencia: valor = 21, índice = 2
Última coincidencia: valor = 21, índice = 98
findAll: índices = [ 2 8 15 20 26 34 38 46 50 58 66 74 82 90 98 ]
Secuencial con centinela: encontrado = 21
Búsqueda binaria: índice = 74
```

LISTA ENLAZADA

- colección: 100 registros de notas
- clave/predicado: [82, 85, 88]
- método y salida:

```
--------------------------------------------------
           DEMO CON LISTA ENLAZADA SIMPLE
--------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 82,85,88
Primera coincidencia de 82: Node {value = 82}
Última coincidencia de 82: Node {value = 82}
Primera coincidencia de 85: Node {value = 85}
Última coincidencia de 85: Node {value = 85}
Primera coincidencia de 88: Node {value = 88}
Última coincidencia de 88: Node {value = 88}
findAll (valor < 20):
(vacío)
--------------------------------------------------
```

- colección: 100 registros de edades
- clave/predicado: [15, 20, 21]
- método y salida:

```
--------------------------------------------------
           DEMO CON LISTA ENLAZADA SIMPLE
--------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 15,20,21
Primera coincidencia no encontrada para 15
Última coincidencia no encontrada para 15
Primera coincidencia de 20: Node {value = 20}
Última coincidencia de 20: Node {value = 20}
Primera coincidencia de 21: Node {value = 21}
Última coincidencia de 21: Node {value = 21}
findAll (valor < 20):
Node {value = 19}
Node {value = 18}
Node {value = 19}
Node {value = 18}
Node {value = 19}
Node {value = 18}
```

inventario_500_inverso.csv

ARRAY

- colección: 500 registros de stock
- método y salida:

```
Buscando clave: 486
--------------------------------------------------
Primera coincidencia: valor = 486, índice = 14
Última coincidencia: valor = 486, índice = 14
findAll: índices = [ 14 ]
Secuencial con centinela: encontrado = 486
Búsqueda binaria: índice = 485

Buscando clave: 440
--------------------------------------------------
Primera coincidencia: valor = 440, índice = 60
Última coincidencia: valor = 440, índice = 60
findAll: índices = [ 60 ]
Secuencial con centinela: encontrado = 440
Búsqueda binaria: índice = 439
```

LISTA ENLAZADA

- colección: 500 registros de stock
- método y salida:

```
--------------------------------------------------
            DEMO CON LISTA ENLAZADA SIMPLE
--------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 486,440
Primera coincidencia de 486: Node {value = 486}
Última coincidencia de 486: Node {value = 486}
Primera coincidencia de 440: Node {value = 440}
Última coincidencia de 440: Node {value = 440}
findAll (valor < 20):
Node {value = 19}
Node {value = 18}
Node {value = 17}
Node {value = 16}
Node {value = 15}
Node {value = 14}
Node {value = 13}
Node {value = 12}
Node {value = 11}
Node {value = 10}
Node {value = 9}
Node {value = 8}
Node {value = 7}
Node {value = 6}
Node {value = 5}
Node {value = 4}
Node {value = 3}
Node {value = 2}
Node {value = 1}
--------------------------------------------------
```

pacientes_500.csv

ARRAY

- colección: 500 registros de id y prioridad
- clave/predicado: [3]
- método y salida:

```
144 145 147 148 149 150 153 154 155 160 161 180 184 187 189 190 196 198 199 2
00 204 210 211 212 213 214 215 217 218 223 224 225 227 230 233 242 246 247 25
144 145 147 148 149 150 153 154 155 160 161 180 184 187 189 190 196 198 199 2
00 204 210 211 212 213 214 215 217 218 223 224 225 227 230 233 242 246 247 25
0 252 259 260 271 273 277 278 280 281 282 283 285 288 289 295 299 300 302 308
 311 318 319 321 322 328 334 339 340 342 343 345 352 355 356 360 365 369 370
379 381 385 386 387 392 393 397 398 399 402 403 404 406 407 411 412 419 420 4
27 433 434 436 440 443 444 445 446 448 449 451 456 457 460 467 468 471 474 47
8 483 484 492 495 497 498 499 ]
Secuencial con centinela: encontrado = 3
Búsqueda binaria: índice = 374
--------------------------------------------------
```

LISTA ENLAZADA

- colección: 500 registros de id y prioridad
- método y salida:

```
------------------------------------------------------
            DEMO CON LISTA ENLAZADA SIMPLE
------------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 3
Primera coincidencia de 3: Node {value = 3}
Última coincidencia de 3: Node {value = 3}
findAll (valor < 20):
Node {value = 2}
Node {value = 3}
Node {value = 1}
Node {value = 1}
Node {value = 1}
```

productos_inventario.csv

ARRAY

- colección: 100 registros de id
- clave/predicado: [1, 50, 23]
- método y salida:

```
Buscando clave: 1
------------------------------------------------------
Primera coincidencia: valor = 1, índice = 0
Última coincidencia: valor = 1, índice = 0
findAll: índices = [ 0 ]
Secuencial con centinela: encontrado = 1
Búsqueda binaria: índice = 0

Buscando clave: 50
------------------------------------------------------
Primera coincidencia: valor = 50, índice = 49
Última coincidencia: valor = 50, índice = 49
findAll: índices = [ 49 ]
Secuencial con centinela: encontrado = 50
Búsqueda binaria: índice = 49

Buscando clave: 23
------------------------------------------------------
Primera coincidencia: valor = 23, índice = 22
Última coincidencia: valor = 23, índice = 22
findAll: índices = [ 22 ]
Secuencial con centinela: encontrado = 23
Búsqueda binaria: índice = 22
------------------------------------------------------
```

- colección: 100 registros de stock
- clave/predicado: [1, 50, 23]
- método y salida:

```
Buscando clave: 1
-----------------------------------------------------
Primera coincidencia no encontrada
Última coincidencia no encontrada
findAll vacío
Secuencial con centinela no encontrado
Búsqueda binaria no encontrado

Buscando clave: 50
-----------------------------------------------------
Primera coincidencia no encontrada
Última coincidencia no encontrada
findAll vacío
Secuencial con centinela no encontrado
Búsqueda binaria no encontrado

Buscando clave: 23
-----------------------------------------------------
Primera coincidencia: valor = 23, índice = 2
Última coincidencia: valor = 23, índice = 84
findAll: índices = [ 2 84 ]
Secuencial con centinela: encontrado = 23
Búsqueda binaria: índice = 36
-----------------------------------------------------
```

LISTA ENLAZADA

- colección: 100 registros de id
- clave/predicado: [1, 50, 23]
- método y salida:

```
-----------------------------------------------------
           DEMO CON LISTA ENLAZADA SIMPLE
-----------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 1,50,23
Primera coincidencia de 1: Node {value = 1}
Última coincidencia de 1: Node {value = 1}
Primera coincidencia de 50: Node {value = 50}
Última coincidencia de 50: Node {value = 50}
Primera coincidencia de 23: Node {value = 23}
Última coincidencia de 23: Node {value = 23}
findAll (valor < 20):
Node {value = 1}
Node {value = 2}
Node {value = 3}
Node {value = 4}
Node {value = 5}
Node {value = 6}
```

- colección: 100 registros de stock
- clave/predicado: [1, 50, 23]
- método y salida:

```
--------------------------------------------------
           DEMO CON LISTA ENLAZADA SIMPLE
--------------------------------------------------
Ingrese claves a buscar separadas por comas (ENTER para usar [3]): 1,50,23
Primera coincidencia no encontrada para 1
Última coincidencia no encontrada para 1
Primera coincidencia no encontrada para 50
Última coincidencia no encontrada para 50
Primera coincidencia de 23: Node {value = 23}
Última coincidencia de 23: Node {value = 23}
findAll (valor < 20):
Node {value = 15}
Node {value = 8}
Node {value = 12}
Node {value = 9}
Node {value = 5}
Node {value = 3}
Node {value = 14}
Node {value = 11}
Node {value = 7}
Node {value = 6}
Node {value = 18}
Node {value = 12}
Node {value = 8}
Node {value = 6}
Node {value = 4}
```

7. Preguntas de Control

- **¿Por qué la binaria no sirve para SLL?**

  Esto se debe a que no se puede acceder al elemento que se encuentra en medio y por ende es necesario recorrer nodo por nodo , con esto se pierde la ventaja que proporciona la búsqueda binaria de dividir a la mitad la lista enlazada.

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

  Esto se debe a que como ya se encontró lo que se estaba buscando , se debe retornar inmediatamente caso contrario seguir en esa búsqueda sería ineficiente.

- **¿Qué garantiza la correctitud del centinela?**

  Primero hay que entender que el centinela consiste en poner la clave que tenemos al final, entonces, al hacer esto se está garantizando que el bucle siempre va a encontrar algo, de esta forma eliminamos la verificación de los límites que hay en cada iteración.

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

  En la primera : cuando se encuentre , seguir buscando hacia la izquierda haciendo uso de la instrucción lógica: high = mid - 1.

  Para la última: hay que seguir buscando hacia el lado derecho, usando la siguiente instrucción lógica: low = mid + 1.

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

Caso Borde 1:

Se realizó una búsqueda de un número que no existía en el arreglo utilizando el algoritmo de Búsqueda Secuencial con Centinela.

Entonces, en la primera versión, el método recibía un parámetro extra centinel. Al buscar una clave que no estaba en el arreglo, el bucle while continuaba indefinidamente o desbordaba el índice (ArrayIndexOutOfBoundsException) porque el valor colocado al final no coincidía con la condición de parada esperada por la lógica del bucle.

En su solución, se refactoriza el método en ArrayUtils para eliminar el parámetro externo. Ahora, el algoritmo coloca explícitamente la clave buscada (key) en la posición n-1. Esto garantiza matemáticamente que el bucle termine. Adicionalmente, se agregó la validación final para distinguir si el índice se detuvo en un dato real o en el centinela artificial.

Caso Borde 2:

Primero se intentó ejecutar la *Búsqueda Binaria* sobre la columna "Edad" del archivo estudiantes_notas.csv buscando el valor 20, el cual existía visualmente en el archivo.

El algoritmo retornaba repetidamente NoSuchElementException. Esto sucedía porque el archivo CSV se lee en el orden original (desordenado), violando la precondición fundamental de la búsqueda binaria.

Para su solución En la clase SearchDemo, se implementó una rutina de preparación que:

1. Crea una copia del arreglo original (Arrays.copyOf).
2. Ordena dicha copia utilizando SelectionSort.
3. Ejecuta la búsqueda binaria sobre la copia ordenada. Esto asegura resultados correctos sin alterar el orden de los datos originales presentados al usuario.

8. Conclusiones

1) La práctica demostró que cada algoritmo de búsqueda tiene un contexto específico en cuanto a su uso , comenzando desde la búsqueda secuencial que funciona en cualquier colección y permite usar predicados personalizables, siendo ideal para datos no ordenados o listas enlazadas. Por otra parte, la búsqueda binaria, aunque es más rápida, requiere datos ordenados y solo funciona en arreglos con acceso directo por índice.
2) La técnica del centinela redujo las comparaciones al eliminar la verificación de límites en cada iteración, con esto se puede concluir que los resultados confirman que no existe un algoritmo superior en todos los casos, sino que la elección correcta depende de la estructura de datos disponible y si está ordenada.
3) En el proceso de resolución de la actividad propuesta fue necesaria la división clara de tareas de los integrantes del equipo, lo que permite trabajar eficientemente de manera paralela y enfocada, para reducir el tiempo de desarrollo y así cumplir el plazo establecido.

9. Evaluación

Criterio	4 – Excelente	3 – Bueno	2 – Básico	1 – Insuficiente	Pts
Secuencial (first/last/findAll)	Correctos; manejan bordes; código claro	Detalles menores	Parcial	No funcional	3.0
Centinela (arrays)	Implementado y explicado; comparación de comparaciones	Implementado	Parcial	No	2.0
Binaria (arrays)	Correcta; cuidado con mid; precondición validada	Detalles menores	Parcial	No	2.5
Evidencias (tabla/README)	Completas y reproducibles	Aceptables	Escasas	Nulas	1.5
Calidad de código	Organización y nombres adecuados	Aceptable	Pobre	Deficiente	1.0

9. Bibliografía

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

[1] 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.

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