Guía de Actividades Práctico Experimentales Nro. 007

1. Datos Generales

Nombre del estudiante(s)	- Alex Sigcho - Ivett Zaragocin - Emilio Flores - Yober Gaona - Marco Orozco
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 (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: cuándo conviene secuencial vs binaria; centinela en "no encontrado".
- Completar README e informe con evidencias y decisiones.

6. Resultados esperados:

- **Tabla (o CSV) con casos: colección, clave/predicado, método, salida.**

Colección / Dataset	Clave / Predicado	Método Aplicado	Resultado Esperado	Resultado Obtenido
Array A [8, 3, 6, 3, 9, 7, 2, 3]	Key: 3	Paso1.indexOfFirst	1 (Primer índice)	1
Array D [] (Vacío)	Key: 42	Paso1.indexOfFirst	-1 (No existe)	-1
SLL 3 -> 1 -> 3 -> 2	Key: 3	Paso1.findFirst	Node{3} (Ref. al 1ro)	Node{3}
Array A [8, 3, 6, 3, 9, 7, 2, 3]	Key: 3	Paso2.lastOccurrenceArray	7 (Último índice)	7
SLL 3 -> 1 -> 3 -> 2	Key: 3	Paso2.findLast	Node{3} (Ref. al 2do)	Node{3}
Array A [8, 3, 6, 3, 9, 7, 2, 3]	Pred: x % 2 == 0	Paso3.findAll	[0, 2, 6] (Índices)	[0, 2, 6]
SLL 3 -> 1 -> 3 -> 2	Pred: val < 3	Paso3.findAll	[Node{1}, Node{2}]	[Node{1}, Node{2}]
Array A [8, 3, 6, 3, 9, 7, 2, 3]	Key: 7	Paso4.searchCentinel	5 (Índice real)	5

Array A [8, 3, 6, 3, 9, 7, 2, 3]	Key: 42 (No está)	Paso4.searchCentinel	-1 (Centinela detectado)	-1
Array B [1, 2, 3, 5, 7, 9, 11, 15]	Key: 7	Paso5.binarySearch	4 (Índice central)	4
Array C [2, 2, 2, 2, 2]	Key: 2	Paso5.binarySearchFirst	0 (Primero de varios)	0
Array B [1, 2, 3, 5, 7, 9, 11, 15]	Key: 5	Paso5.binarySearchFirst	3 (Índice único)	3

Explicación de la tabla:

Array A: Es el dataset desordenado principal para pruebas secuenciales.

Array B: Es el dataset ordenado obligatorio para probar la búsqueda binaria.

Array C: Es un caso borde especial (todos iguales) para verificar que binarySearchFirst no se detiene en el medio, sino que retrocede hasta el inicio.

SLL: Representa la Lista Simplemente Enlazada (3 -> 1 -> 3 -> 2).

- **README: cómo compilar/ejecutar; casos bordes; notas sobre precondiciones**

Link del GitHub:

https://github.com/NullDistortion/Taller-7-Busqueda-e-Implementacion

- **Capturas de ejecución**

```
1. Primera Ocurrencia (indexOfFirst)
paso 1: indexOfFirst (Array, int)
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], key=7
resultado: Índice: 5
paso 1: indexOfFirst (Array, int)
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], key=3
resultado: Índice: 1
paso 1: indexOfFirst (Array, int)
dataset usado: D [] (Vacío), key=5
resultado: Índice: -1
paso 1: indexOfFirst (SLL, int)
dataset usado: SLL 3->1->3->2, key=3
resultado: Índice: 0
```

```
2. Última Ocurrencia
paso 2: lastOccurrenceArray (Array)
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], key=3
resultado: Índice: 7
paso 2: lastOccurrenceArray (Array)
dataset usado: C [2, 2, 2, 2, 2] (Duplicados), key=2
resultado: Índice: 4
paso 2: indexOfLast (SLL)
dataset usado: SLL 3->1->3->2, key=3
resultado: Índice: 2
```

```
3. findAll por Predicado
paso 3: findAll (Array)
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], Predicado: 'isPair()'
resultado: Índices: [0, 2, 6]
paso 3: findAll (Array)
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], Predicado: 'equals(3)'
resultado: Índices: [1, 3, 7]
paso 3: findAllIndexes (SLL)
dataset usado: SLL 3->1->3->2, Predicado: 'val < 3'
resultado: Índices: [1, 3]
```

```
4. Secuencial con Centinela
paso 4: searchCentinel
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], key=9
resultado: Índice: 4
paso 4: searchCentinel
dataset usado: A [8, 3, 6, 3, 9, 7, 2, 3], key=42
resultado: Índice: -1
paso 4: searchCentinel
dataset usado: D [] (Vacío), key=5
resultado: Índice: -1
```

```
5. Búsqueda Binaria
paso 5: binarySearch (Estándar)
dataset usado: B [1, 2, 3, 5, 7, 9, 11, 15] (Ordenado), key=7
resultado: Índice: 4
paso 5: binarySearch (Estándar)
dataset usado: B [1, 2, 3, 5, 7, 9, 11, 15] (Ordenado), key=42
resultado: Índice: -1
paso 5: binarySearchFirst (Lower Bound)
dataset usado: C [2, 2, 2, 2, 2] (Duplicados), key=2
resultado: Primer Índice (Lower Bound): 0
```

7. Preguntas de Control:

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

  Porque la binaria divide el los datos dependiendo de lo que buscará y en una SLL no se puede dividir sus datos ya que estos se encuentran enlazados unos con otros y si se dividen se perdería el la información del nodo siguiente.

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

  Porque va comparando el dato de la posición en la que se encuentra con el dato que buscamos y si es igual este devolverá la posición del dato que cumpla la condición.

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

  Se garantiza al colocar el valor que se está buscando al final como centinela y si el índice es menor que la posición del centinela el dato si existe y si su índice es igual a la posición del centinela

significa que el dato no existe.

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

```
if (a[mid] == key) {
    result = mid;     // Guardamos el índice encontrado
    high = mid - 1;   // Seguimos buscando a la izquierda por si hay otro antes
```

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

  Cuando el vector está vacío

  Cuando solo hay duplicados

```
/**
 * Dataset C: Elementos duplicados
 */
public static int[] getArrayC() {  no usages
    return new int[]{2, 2, 2, 2, 2};
}

/**
 * Dataset D: Caso borde: Vacio
 */
public static int[] getArrayD() {  no usages
    return new int[]{};
}
```

8. Discusión

1. ¿Búsqueda Secuencial o Búsqueda Binaria?

La intuición inicial sugiere que la Búsqueda Binaria es siempre superior por su complejidad frente a la lineal de la Búsqueda Secuencial. Sin embargo, las pruebas revelaron matices importantes:

Dependencia del Orden: La Búsqueda Binaria falló o dio resultados erróneos en el Dataset A (desordenado). Esto confirma que su uso está restringido estrictamente a datos previamente ordenados. Si los datos cambian frecuentemente, el costo de mantener el orden puede superar el beneficio de la búsqueda rápida.

Restricción de Estructura: No pudimos aplicar Búsqueda Binaria eficientemente en la Lista Enlazada. La necesidad de acceder al elemento medio es instantánea en arreglos, pero costosa en listas, anulando la ventaja de velocidad.

Conclusión: La Búsqueda Binaria es ideal para arreglos estáticos y ordenados. Para listas enlazadas o datos desordenados pequeños, la Búsqueda Secuencial sigue siendo la opción viable.

2. *El Centinela en el Caso "No Encontrado"*

Analizamos el comportamiento de la Búsqueda con Centinela cuando el elemento buscado NO existe

- Al salir del bucle, el algoritmo debe discernir si encontró el dato real o el centinela "falso". Esto se logra comparando el índice final (i < n-1) o verificando el valor original que fue respaldado.
- En el peor caso (elemento no encontrado), la búsqueda con centinela realiza N comparaciones de datos + 1 verificación final. La búsqueda clásica realizaría 2N comparaciones (límite + dato).

Conclusión: El centinela ofrece una mejora de rendimiento constante (aprox. 50% menos comparaciones de control) a cambio de una operación de escritura en memoria, lo cual es ventajoso en arreglos grandes.

9. Conclusiones

Gracias a la práctica realizada se pudo analizar la eficiencia y aplicabilidad de los algoritmos de búsqueda como:

- Se confirmó que la Búsqueda Binaria es, por mucho, el algoritmo más eficiente para encontrar elementos en grandes volúmenes de datos. Sin embargo, su dependencia estricta del orden la hace inaplicable en escenarios donde los datos cambian constantemente y el costo de reordenar supera el beneficio de la búsqueda rápida
- La Búsqueda Secuencial demostró ser la única opción viable para estructuras no indexadas como las Listas Enlazadas (SLL) y para arreglos pequeños o desordenados donde la simplicidad de implementación es prioritaria
- La implementación del centinela reduce significativamente el overhead de la búsqueda secuencial y representa una optimización de bajo nivel valiosa para sistemas críticos
- Las pruebas unitarias en DatasetsPrueba revelaron que la robustez de un algoritmo de búsqueda no se mide solo cuando encuentra el dato, sino cuando no lo encuentra o cuando la estructura está vacía. La correcta gestión de null y arreglos vacíos (length == 0) es indispensable para evitar fallos en el tiempo de ejecución

10. 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] P. W. Bible and L. Moser, An Open Guide to Data Structures and Algorithms. PALNI Open Press, 2023.

[2] OpenDSA Project, "Searching and Sorting Modules," Virginia Tech, 2021–2024 (REA con visualizaciones).

[3] Oracle, "Java SE 17–21 Documentation: Arrays.binarySearch, Comparator y patrones de búsqueda," 2021–2025.

10. Elaboración y Aprobación

Elaborado por	Andrés R Navas Castellanos **Docente**	
Revisado por **Solo si es realizado en laboratorios**	Luis Sinche **Técnico Docente**	No Aplica
Aprobado por	Edison L Coronel Romero **Director de Carrera**	