

Guía de Actividades Práctico-Experimentales Nro. 007

1. Datos Generales

Integrantes	Miguel Alejandro Armas Ordóñez Richard Vicente Cajas Riofrio Mateo Fabian Silva Aguilar
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: $mid = low + (high - low) / 2$, precondition 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: $3 \rightarrow 1 \rightarrow 3 \rightarrow 2$, claves: 3 (primera/última) y predicado `val < 3`.
 - Registrar índices/nodos esperados y observados.
 - Evidencias: tabla con entradas, método y salida.



unl

Universidad
Nacional
de Loja

6. Resultados esperados:

En estas pruebas devuelve siempre la primera y última ocurrencia, al haber valores únicos la primera y la última ocurrencia será la misma

```
=====
PRUEBAS CON ARREGLOS
=====

Devolverá el índice del valor, en caso de no existir devolverá -1
Dataset: Arreglo A (Desordenado)
Contenido: [ 8 3 6 3 9 ]
Clave 7 -> First: -1 | Last: -1
Clave 5 -> First: -1 | Last: -1
Clave 2 -> First: -1 | Last: -1
Clave 42 -> First: -1 | Last: -1
```

Como ninguna de las claves coincide dentro del arreglo devuelve -1 (no existe)

```
Devolverá el índice del valor, en caso de no existir devolverá -1
Dataset: Arreglo B (Inverso)
Contenido: [ 5 4 3 2 1 ]
Clave 7 -> First: -1 | Last: -1
Clave 5 -> First: 0 | Last: 0
Clave 2 -> First: 3 | Last: 3
Clave 42 -> First: -1 | Last: -1
```

Cómo los valores de 5 y 2 existen dentro del arreglo devuelve la el índice de donde existen esos valores, y -1 para los valores que no existen.

```
Devolverá el índice del valor, en caso de no existir devolverá -1
Dataset: Arreglo C (Ordenado)
Contenido: [ 1 2 3 4 5 ]
Clave 7 -> First: -1 | Last: -1
Clave 5 -> First: 4 | Last: 4
Clave 2 -> First: 1 | Last: 1
Clave 42 -> First: -1 | Last: -1
```

Cómo los valores de 5 y 2 existen dentro del arreglo devuelve la el índice de donde existen esos valores, y -1 para los valores que no existen.

```
Devolverá el índice del valor, en caso de no existir devolverá -1
Dataset: Arreglo D (Duplicados)
Contenido: [ 2 2 2 2 ]
Clave 7 -> First: -1 | Last: -1
Clave 5 -> First: -1 | Last: -1
Clave 2 -> First: 0 | Last: 3
Clave 42 -> First: -1 | Last: -1
```

Como 2 está duplicado dentro del arreglo, la primera y la ultima ocurrencia van a diferir y devuelve los índices de la primera y la última ocurrencia



```
--- Prueba Búsqueda Binaria (Solo en C - Ordenado) ---  
Contenido: [ 1 2 3 4 5 ]  
BinarySearch de 7 en C: -1  
BinarySearch de 5 en C: 4  
BinarySearch de 2 en C: 1  
BinarySearch de 42 en C: -1
```

La búsqueda binaria se realizó solo en el arreglo ordenado porque es prerequisite para poder realizar esta búsqueda.

VERSUS DE BUSQUEDA CON Y SIN CENTINELA

```
=====
VERSUS: SECUENCIAL vs CENTINELA (Trazas)
=====
Contenido: [ 8 3 6 3 9 ]
--- Buscando clave: 9 en A ---
[Secuencial] Comparaciones Totales (Bucle + If): 10
[Centinela] Comparaciones realizadas: 5
--> Resultado índices: Secuencial=4, Centinela=4
```

Para la primera comparación usamos el primer arreglo, para evaluar uno de los "peores casos" donde el valor se encuentra en el final del arreglo, comparando la cantidad de comparaciones que se hace en cada tipo de búsqueda.

```
=====
VERSUS: SECUENCIAL vs CENTINELA (Trazas) (No existe el valor)
=====
Contenido: [ 8 3 6 3 9 ]
--- Buscando clave: 5 en A ---
[Secuencial] Comparaciones Totales (Bucle + If): 11
[Centinela] Comparaciones realizadas: 6
--> Resultado índices: Secuencial=-1, Centinela=5
Como el index devuelto es igual al tamaño del arreglo: 5, el elemento no está en el arreglo
```

Para esta segunda comparación usamos el primer arreglo pero buscando un valor que no existe dentro del arreglo, como podemos ver las comparaciones entre los métodos de búsqueda con y sin centinela, y dependiendo lo que se devuelva damos una conclusión, si el índice devuelto es igual al tamaño del arreglo significa que está en la posición extra que añadimos, por ende no está contenido en el arreglo original.

```
=====
PRUEBAS CON LISTA ENLAZADA (SLL)
=====

Lista: [3 => 1 => 3 => 2]
First Match (3): Hash 1442407170
Last Match (3): Hash 1028566121
FindAll (menores a 3)(<3): 1 =>2
```

Para la lista enlazada definimos una lista 3->1->3->2 entonces buscamos todas las coincidencias de valores menores a 3.

PRUEBAS TODAS LAS COINCIDENCIAS ARREGLO

Contenido: [1 2 3 3 4 5 6 3 7 8 9 3]
FindAll (iguales a 3)(=3): Todos los índices donde hay coincidencias
[2, 3, 7, 11]

Para devolver todas las coincidencias creamos un arreglo con coincidencias y devolvemos los índices de todos los valores donde se encontraron las coincidencias.

Cierre

- Discusión: ¿Cuándo conviene Secuencial vs. Binaria?

La búsqueda secuencial es mejor cuando el arreglo está desordenado, la cantidad de datos es relativamente pequeño (máximo unos 100), y en listas enlazadas, ya que no es posible acceder de otra forma de acceder que no sea recorrer cada uno de los nodos previos.

Para la búsqueda binaria es muy recomendada cuando el arreglo cumpla con el prerequisite, el cual es que el arreglo esté ordenado, ya que reduce significativamente el tiempo de búsqueda, y no es posible hacer esta búsqueda en listas enlazadas, porque no hay forma de acceder a los índices en listas enlazadas.

El Fenómeno del Centinela en el caso "No Encontrado"

En un bucle for normal de búsqueda secuencial, la CPU evalúa dos condiciones en cada vuelta:

1. ¿Sigo dentro del arreglo? ($i < n$)
2. ¿Es el dato lo que busco? ($a[i] == \text{key}$)

La Mejora del Centinela:

Al colocar la key (clave) artificialmente en la posición n (al final), garantizamos que el número siempre será encontrado, ya sea el real o el falso (centinela). Esto nos permite eliminar la pregunta 1 ($i < n$) del bucle.

Es donde más brilla la optimización del Centinela es cuando los arreglos son de gran cantidad ya que las comparaciones que se harán son 2 por cada índice del arreglo, a diferencia de la búsqueda con centinela donde el número de comparaciones se reduce hasta la mitad.

7. Preguntas de Control:

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

Porque en las listas enlazadas se está obligado a recorrer todos y cada 1 de los nodos previos para poder acceder o encontrar el nodo deseado, no se puede acceder mediante índices como en los arreglos así que es obligatorio el recorrido.

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

Para finalizar las repeticiones antes, una vez encontrada la coincidencia no tiene sentido seguir recorriendo el arreglo si ya se encontró la coincidencia.

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

La correctitud de la variante centinela no se refiere solo a "menos comparaciones", sino a que el algoritmo siempre termina correctamente sin necesidad de verificar límites en cada iteración. El centinela garantiza que el valor buscado aparecerá (al menos en la última posición), evitando errores de índice y simplificando el bucle.

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

Para la búsqueda binaria con duplicados normalmente te enviaría cualquier coincidencia y finalizaría, pero se lo puede adaptar para que envíe la primero o la última coincidencia de ese grupo de valores duplicados.



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

Cuando la lista enlazada es nula o cuando los arreglos están vacíos.

Cuando en la búsqueda binaria se trabaja con un arreglo desordenado, siempre debe recibir un arreglo ordenado, entonces se lo debe ordenar previamente antes de realizar la búsqueda.

8. 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; precondition 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	 <small>Firmado electrónicamente por:</small> ANDRES ROBERTO NAVAS CASTELLANOS <small>Validar Únicamente con FirmaEC</small>
Revisado por Solo si es realizado en laboratorios	Luis Sinche Técnico Docente	No Aplica
Aprobado por	Edison L Coronel Romero Director de Carrera	 <small>Firmado electrónicamente por:</small> EDISON LEONARDO CORONEL ROMERO <small>Validar Únicamente con FirmaEC</small>