



**UNL**

Universidad  
Nacional  
de Loja  
1859

FEIRNNR - Carrera de Computación

# Guía de Actividades Práctico-Experimentales Nro. 007

## 1. Datos Generales

<b>Asignatura</b>	Estructura de datos
<b>Ciclo</b>	3 A
<b>Unidad</b>	2
<b>Integrantes</b>	<ul style="list-style-type: none"><li>- Cael Alejandro Soto Castillo</li><li>- Cristhian Alexander Dávila Sari</li><li>- Martina Alejandra Maldonado Machuca</li><li>- Fernando Sebastian Patiño Diaz</li></ul>
<b>Resultado de aprendizaje de la unidad</b>	Aplica los métodos de ordenación y búsqueda en la resolución de problemas, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
<b>Título de la Práctica</b>	Búsqueda en Java: Secuencial y Binaria
<b>Nombre del Docente</b>	Andrés Roberto Navas Castellanos
<b>Fecha</b>	Jueves 27 de noviembre
<b>Horario</b>	07h30 – 10h30
<b>Lugar</b>	Aula
<b>Tiempo planificado en el Sílabo</b>	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 DLL).

## 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 indexOffFirst(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”.**

Después de implementar y probar estos métodos de búsqueda, vimos que la búsqueda secuencial es la más adecuada cuando los datos no están ordenados o cuando trabajamos con listas enlazadas, ya que en una SLL no se puede acceder directamente a posiciones específicas. En cambio, la búsqueda binaria sólo conviene en arreglos ordenados, donde sí es posible calcular el índice medio de manera eficiente y obtener un mejor rendimiento. Respecto al centinela, comprobamos que ayuda a reducir comparaciones dentro del bucle, pero requiere verificar al final si el elemento realmente estaba en el arreglo o si sólo se encontró por efecto del centinela. Esta verificación es clave sobre todo cuando la clave no se encuentra, para evitar falsos positivos.

- **Completar README e informe con evidencias y decisiones.**



## 6. Resultados esperados:

- ZIP con src/ (implementaciones y SearchDemo).
- Tabla (o CSV) con casos: colección, clave/predicado, método, salida.
- README (1 pág.): cómo compilar/ejecutar; casos bordes; notas sobre precondiciones.
- (Opcional +) Comparación de comparaciones entre secuencial clásica vs centinela.

### - TABLA CON CASOS:

Key	Dataset	Tipo de búsqueda	Resultado
2	A (mezclado)	Secuencial	1
2	A (mezclado)	Centinela	1
2	A (mezclado)	Primera ocurrencia	1
2	A (mezclado)	Última ocurrencia	99
2	A (mezclado)	Binaria	24
2	B (descendente)	Secuencial	-1
2	B (descendente)	Centinela	-1
2	B (descendente)	Primera ocurrencia	-1
2	B (descendente)	Última ocurrencia	-1
2	B (descendente)	Binaria	-1
2	C (ascendente)	Secuencial	1
2	C (ascendente)	Centinela	1
2	C (ascendente)	Primera ocurrencia	1
2	C (ascendente)	Última ocurrencia	1
2	C (ascendente)	Binaria	1
2	D (casi ordenado)	Secuencial	91
2	D (casi ordenado)	Centinela	91
2	D (casi ordenado)	Primera ocurrencia	91
2	D (casi ordenado)	Última ocurrencia	91
2	D (casi ordenado)	Binaria	-1
2	A (mezclado)	SLL todas las ocurrencias	[1,2,8,12,13,16,17,18,24,25,29,33,37,38,42,51,54,59,71,75,78,87,94,99]
2	A (mezclado)	SLL primera ocurrencia	1
2	A (mezclado)	SLL última ocurrencia	99
2	B (descendente)	SLL todas las ocurrencias	[]
2	B (descendente)	SLL primera ocurrencia	-1
2	B (descendente)	SLL última ocurrencia	-1
2	C (ascendente)	SLL todas las ocurrencias	[1]
2	C (ascendente)	SLL primera ocurrencia	1
2	C (ascendente)	SLL última ocurrencia	1
2	D (casi ordenado)	SLL todas las ocurrencias	[91]
2	D (casi ordenado)	SLL primera ocurrencia	91
2	D (casi ordenado)	SLL última ocurrencia	91
5	A (mezclado)	Secuencial	-1
5	A (mezclado)	Centinela	-1
5	A (mezclado)	Primera ocurrencia	-1
5	A (mezclado)	Última ocurrencia	-1
5	A (mezclado)	Binaria	-1
5	B (descendente)	Secuencial	-1
5	B (descendente)	Centinela	-1
5	B (descendente)	Primera ocurrencia	-1
5	B (descendente)	Última ocurrencia	-1
5	B (descendente)	Binaria	-1
5	C (ascendente)	Secuencial	4
5	C (ascendente)	Centinela	4
5	C (ascendente)	Primera ocurrencia	4
5	C (ascendente)	Última ocurrencia	4
5	C (ascendente)	Binaria	4
5	D (casi ordenado)	Secuencial	70
5	D (casi ordenado)	Centinela	70
5	D (casi ordenado)	Primera ocurrencia	70
5	D (casi ordenado)	Última ocurrencia	70
5	D (casi ordenado)	Binaria	-1
5	A (mezclado)	SLL todas las ocurrencias	[]
5	A (mezclado)	SLL primera ocurrencia	-1
5	A (mezclado)	SLL última ocurrencia	-1
5	B (descendente)	SLL todas las ocurrencias	[]
5	B (descendente)	SLL primera ocurrencia	-1
5	B (descendente)	SLL última ocurrencia	-1
5	C (ascendente)	SLL todas las ocurrencias	[4]
5	C (ascendente)	SLL primera ocurrencia	4
5	C (ascendente)	SLL última ocurrencia	4
5	D (casi ordenado)	SLL todas las ocurrencias	[70]
5	D (casi ordenado)	SLL primera ocurrencia	70
5	D (casi ordenado)	SLL última ocurrencia	70



**UNL**

Universidad  
Nacional  
de Loja

1859

## FEIRNNR - Carrera de Computación

```
== DATASET A (mezclado) ==
[ 1 2 2 3 3 1 3 1 2 1 3 3 2 2 3 3 2 2 2 1 3 3 1 3 2 2 1 3 1 2 1 1 3 2 3 3 3 2 2 1 3 1 2 3 1 3 3 3 1 3 3 2 3 1 2 1 3 1 1 2
== DATASET B (descendente) ==
[ 500 499 498 497 496 495 494 493 492 491 490 489 488 487 486 485 484 483 482 481 480 479 478 477 476 475 474 473 472 471
== DATASET C (ascendente) ==
[ 1 2 3 4 5 6 7 8 9 10 ]
== DATASET D (casi ordenado) ==
[ 66 65 14 98 64 19 76 25 51 38 36 74 18 13 57 83 63 43 41 71 47 27 32 79 6 40 55 95 80 67 31 1 82 22 33 94 77 78 12 69 2
== LISTA A ==
[ 1 → 2 → 2 → 3 → 3 → 1 → 3 → 1 → 2 → 1 → 3 → 3 → 2 → 2 → 3 → 2 → 2 → 1 → 3 → 3 → 1 → 3 → 2 → 2 → 1 → 3 → 1 → 2 →
== LISTA B ==
[ 500 → 499 → 498 → 497 → 496 → 495 → 494 → 493 → 492 → 491 → 490 → 489 → 488 → 487 → 486 → 485 → 484 → 483 → 482 → 481 →
== LISTA C ==
[ 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 ]
== LISTA D ==
[ 66 → 65 → 14 → 98 → 64 → 19 → 76 → 25 → 51 → 38 → 36 → 74 → 18 → 13 → 57 → 83 → 63 → 43 → 41 → 71 → 47 → 27 → 32 → 79 →
```

Ingrese la clave (key) que desea buscar:

Ingrese la clave (key) que desea buscar: 2

=====

### BÚSQUEDAS EN ARRAY

=====

=====Dataset A=====

```
Secuencial A = 1
Centinela A = 1
Primera ocurrencia A = 1
Última ocurrencia A = 99
Binaria A = 24
```

=====Dataset B=====

```
Secuencial B = -1
Centinela B = -1
Primera ocurrencia B = -1
Última ocurrencia B = -1
Binaria B = -1
```

=====Dataset C=====

```
Secuencial C = 1
Centinela C = 1
Primera ocurrencia C = 1
Última ocurrencia C = 1
Binaria C = 1
```

=====Dataset D=====

```
Secuencial D = 91
Centinela D = 91
Primera ocurrencia D = 91
Última ocurrencia D = 91
Binaria D = -1
```



**UNL**

Universidad  
Nacional  
de Loja

1859

## FEIRNNR - Carrera de Computación

```
=====
BÚSQUEDAS EN SLL
=====

SLL búsqueda A = [1, 2, 8, 12, 13, 16, 17, 18, 24, 25, 29, 33, 37, 38, 42, 51, 54, 59, 71, 75, 78, 87, 94, 99]
SLL primera A = 1
SLL última A = 99

SLL búsqueda B = []
SLL primera B = -1
SLL última B = -1

SLL búsqueda C = [1]
SLL primera C = 1
SLL última C = 1

SLL búsqueda D = [91]
SLL primera D = 91
SLL última D = 91
Desea realizar otra búsqueda (key) (s/n):
```

```
Ingrese la clave (key) que desea buscar: 5
```

```
=====
BÚSQUEDAS EN ARRAY
=====

=====Dataset A=====
Secuencial A = -1
Centinela A = -1
Primera ocurrencia A = -1
Última ocurrencia A = -1
Binaria A = -1

=====Dataset B=====
Secuencial B = -1
Centinela B = -1
Primera ocurrencia B = -1
Última ocurrencia B = -1
Binaria B = -1

=====Dataset C=====
Secuencial C = 4
Centinela C = 4
Primera ocurrencia C = 4
Última ocurrencia C = 4
Binaria C = 4

=====Dataset D=====
Secuencial D = 70
Centinela D = 70
Primera ocurrencia D = 70
Última ocurrencia D = 70
Binaria D = -1
```



**UNL**

Universidad  
Nacional  
de Loja

1859

FEIRNNR - Carrera de Computación

```
=====
BÚSQUEDAS EN SLL
=====

SLL búsqueda A = []
SLL primera ocurrencia A = -1
SLL última ocurrencia A = -1

SLL búsqueda total B = []
SLL primera ocurrencia B = -1
SLL última ocurrencia B = -1

SLL búsqueda total C = [4]
SLL primera ocurrencia C = 4
SLL última ocurrencia C = 4

SLL búsqueda total D = [70]
SLL primera ocurrencia D = 70
SLL última ocurrencia D = 70
Desea realizar otra busqueda (key) (s/n):
```

LINK REPOSITORIO: <https://github.com/C-ael/Taller-7-Implementacion-algoritmos-de-busqueda>

## 7. Preguntas de Control:

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

La búsqueda binaria no es adecuada para una lista simplemente enlazada aunque la lista esté ordenada, porque este tipo de estructura no permite acceder directamente a un índice. En un arreglo, calcular la posición del medio y acceder al valor cuesta  $O(1)$ , pero en una SLL llegar al nodo "mid" implica recorrer uno por uno todos los nodos anteriores, lo cual cuesta  $O(n)$ . Como la búsqueda binaria necesita saltar varias veces entre mitades distintas, el recorrido termina siendo repetitivo y más costoso que simplemente recorrer la lista una sola vez de inicio a fin. Por eso, aunque la lista esté ordenada, la operación binaria pierde completamente su ventaja y deja de ser útil.

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

En la primera ocurrencia, el método retorna apenas encuentra la clave porque en ese punto ya se garantiza que no existe una aparición previa del dato. Como el recorrido es lineal desde el inicio, la primera coincidencia encontrada es, la más cercana al comienzo. Seguir avanzando no cambia el resultado y solo agrega trabajo innecesario, por eso se corta el ciclo inmediatamente.

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

La correctitud de la variante centinela se garantiza por dos aspectos principales: primero, al colocar el elemento buscado al final del arreglo, nos aseguramos de que siempre vamos a encontrar algo (o el elemento real o el centinela), lo que elimina la necesidad de verificar los límites en cada iteración. Segundo, después de terminar la búsqueda, verificamos si el elemento encontrado es el centinela o el elemento real, y restauramos el arreglo original si fue modificado. Esto asegura que el resultado sea válido y que el arreglo quede en su estado original.



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

Para adaptar la búsqueda binaria con duplicados, implementaría dos variantes:

- **Para la primera ocurrencia:** Después de encontrar una coincidencia, se sigue la búsqueda hacia la izquierda (reduciendo el high) para ubicar la frontera donde empiezan los duplicados.
- **Para la última ocurrencia:** Tras hallar una coincidencia, se desplaza la búsqueda hacia la derecha (aumentando el low) hasta encontrar el punto donde terminan las repeticiones.

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

En nuestras pruebas encontramos dos casos borde que inicialmente causaron problemas:

- El primero fue encontrar CSV que tenían un carácter BOM al inicio o separadores distintos, lo que hacía que los valores se leyieran vacíos o se procesaran incorrectamente, generando arreglos llenos de ceros o datos perdidos. Ese problema obligó a limpiar la línea y a detectar correctamente el separador para evitar fallos de lectura.
- El segundo caso borde ocurrió cuando se implementó inicialmente el método findAll en SLL, ya que la impresión mostraba algo como "[I@5197848c]", lo que revelaba que el método estaba devolviendo y mostrando el arreglo sin formateo o utilizando mal la conversión entre lista e int[], mostrando una referencia de memoria en lugar de los índices reales. Estos casos ayudaron a corregir la lectura de datos y la lógica de conversión de resultados.

## 8. Conclusiones

- La elección del algoritmo depende totalmente del contexto, quedó en claro que no existe un método de búsqueda "mejor" en todos los casos. La búsqueda secuencial es más flexible y funciona con cualquier estructura, mientras que la binaria es mucho más eficiente pero solo sirve con arreglos ordenados.
- Los casos borde hacen la diferencia entre un código bueno y uno robusto, al principio los algoritmos funcionaban bien con casos normales, pero fallaban con arreglos vacíos, un solo elemento o con duplicados. Solo después de probar exhaustivamente estos casos extremos se pudo crear código confiable.
- La implementación de la técnica del centinela nos demostró cómo pequeños cambios lógicos (como eliminar una comparación en el bucle) pueden optimizar el código, demostró ser una optimización válida, disminuyendo comparaciones internas, pero también evidenció la necesidad de verificar correctamente el caso "no encontrado" para evitar errores.



**UNL**

Universidad  
Nacional  
de Loja  
1859

FEIRNNR - Carrera de Computación

## 9. Evaluación

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

## 10. 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.

## 11. Elaboración y Aprobación

<b>Elaborado por</b>	Andrés R Navas Castellanos <b>Docente</b>	 Firmado electrónicamente por: <b>ANDRES ROBERTO NAVAS CASTELLANOS</b> Validar únicamente con FirmaBC
<b>Revisado por</b> <b>Solo si es realizado en laboratorios</b>	Luis Sinche <b>Técnico Docente</b>	No Aplica
<b>Aprobado por</b>	Edison L Coronel Romero <b>Director de Carrera</b>	 Firmado electrónicamente por: <b>EDISON LEONARDO CORONEL ROMERO</b> Validar únicamente con FirmaBC