# Guía de Actividades Práctico-Experimentales Nro. 007

## 1. Datos Generales

| | |
|---|---|
| **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 |
| **Estudiantes** | • Anderson Mateo Coello Jaramillo<br>• Daniel Alejandro Saavedra Jaramillo<br>• Ana Cristina Panamito Flores<br>• Royel Ivan Jima Pardo |
| **Link del repositorio** | https://github.com/AndersonC15/taller7-algoritmo-busqueda-estructura-3ciclo-u2 |

## 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:

- 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.

**Búsqueda secuencial clásica**

```
==========================================
=== DEMOSTRACION DE ALGORITMOS DE BUSQUEDA ===
==========================================

--- A. BUSQUEDA SECUENCIAL CLASICA ---
Arreglo A: [1, 3, 7, 7, 9]
  -> Primero(7) esperado 2 | Obtenido: 2
Arreglo D: [7, 5, 2, 42]
  -> Ultimo(7) esperado 0 | Obtenido: 0
  -> Primero(42) esperado -1 | Obtenido: -1
  -> Primero(5) en [] esperado -1 | Obtenido: -1

--- B. BUSQUEDA POR PREDICADO ---
Arreglo D: [7, 5, 2, 42]
  -> Indices Pares esperado [2, 3] | Obtenido: [2, 3]
Arreglo A: [1, 3, 7, 7, 9]
  -> Índices == 7 esperado [2, 3] | Obtenido: [2, 3]
```

**Búsqueda con Centinela**

```
--- C. BUSQUEDA SECUENCIAL CON CENTINELA ---
Arreglo D (original): [7, 5, 2, 42]

Caso 1: Clave 7 (Encontrado)
Valor encontrado en el índice: 0
Comparaciones realizadas: 2
  -> Índice Centinela: 0

Caso 2: Clave 4 (No Encontrado)
Valor NO encontrado.
Comparaciones realizadas: 5
  -> Índice Centinela: -1
```

```
--- D. BUSQUEDA EN LISTA SIMPLE ENLAZADA ---
Lista: [3, 1, 3, 2]
  -> Primero(3): Valor 2
  -> Ultimo(3): Valor 2
  -> Nodos valor < 3: 0 nodos encontrados.
```

## 7. Preguntas de Control:

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

  La búsqueda binaria no es adecuada para una SLL porque requiere acceso aleatorio a los elementos en tiempo constante. En una Lista Enlazada Simple, para llegar al elemento central o cualquier elemento, es necesario recorrer secuencialmente todos los nodos anteriores desde la cabeza, lo que implica un costo. Al hacer esto en cada iteración, la complejidad total de la búsqueda binaria en la SLL aumentaría, haciéndola menos eficiente que una simple búsqueda secuencial

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

  Se retorna tan pronto como se encuentra la clave porque el objetivo de la búsqueda de la primera ocurrencia es encontrar la coincidencia con el menor índice posible (en los arreglos) o el nodo más cercano de la cabeza en la SLL. Dado que la búsqueda secuencial se inicia del principio de la estructura, la primera coincidencia encontrada es, por definición, la primera ocurrencia y detener la ejecución en ese punto ahorra tiempo de procesamiento

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

  La correctitud de la variable centinela se garantiza principalmente por dos pasos:

  - Al colocar la clave búsqueda (key) en la última posición del arreglo, se garantiza que buble while siempre terminará, eliminando el chequeo de límites dentro del bucle
  - Verificación final: Una vez que el bucle termina, se compara el índice final con la posición centinela y el valor original restaurado (último) para determinar si el hallazgo fue real o si se detuvo por la centinela.

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

  Para adaptar la búsqueda binaria en arreglos ordenados y encontrar la primera o la última ocurrencia de una clave duplicada se modifica el movimiento del puntero o indicador para encontrar una coincidencia

  - Primera ocurrencia: Se guarda la posición actual como resultado potencial y se sigue buscando en la mitad izquierda para encontrar una ocurrencia aún más temprana.
  - Última ocurrencia: Se guarda la posición actual como resultado potencial y se sigue buscando en la mitad derecha para encontrar una ocurrencia aun más tardía.

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

  Dos casos borde comunes que detectamos errores de implementación fueron:

  - **Caso Borde 1:** Búsqueda de la última ocurrencia (indexOfLast) en un arreglo vacío ([]). Este caso detecta si el código falla al intentar acceder a a.length-1 sin verificar primero si la longitud del arreglo es cero o si el arreglo es null.
  - **Caso Borde 2:** Búsqueda binaria (binarySearch) en un arreglo de un solo elemento (ej. {7}). Este caso detecta errores en el cálculo inicial del punto medio (mid) o en las condiciones de salida del bucle while(low <= high) cuando low y high son iguales, lo que puede provocar un bucle infinito o un retorno incorrecto.

## 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; 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 |

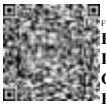
## 9. Bibliografía

[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<br>**Docente** |  <br>Firmado electrónicamente por:<br>**ANDRES ROBERTO NAVAS CASTELLANOS**<br>Validar únicamente con FirmaEC |
| **Revisado por**<br>**Solo si es realizado en laboratorios** | Luis Sinche<br>**Técnico Docente** | No Aplica |
| **Aprobado por** | Edison L Coronel Romero<br>**Director de Carrera** |  <br>Firmado electrónicamente por:<br>**EDISON LEONARDO CORONEL ROMERO**<br>Validar únicamente con FirmaEC |