Guía de Actividades Práctico- Experimentales Nro. 007

1. Datos Generales

Integrantes	Richard Ariel Chamba Santin Katheryn Melissa Contento Maldonado Servio Julian Vega Jimenez Antony José Yaguana Pérez
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 bordes, 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. Conceptos clave

Búsqueda Secuencial

La búsqueda secuencial (o lineal) consiste en recorrer los elementos de una estructura de datos uno por uno hasta encontrar el valor deseado o llegar al final de la estructura.

Características principales:

- No requiere que los datos se encuentren ordenados.
- Puede aplicarse a arreglos y listas simplemente enlazadas (SLL).

Búsqueda Binaria

La búsqueda binaria divide repetidamente todo el conjunto de datos a la mitad, descartando la parte donde no puede encontrarse el elemento buscado.

Características principales:

- Requiere que los datos estén previamente ordenados.
- Se aplica eficientemente sobre arreglos.

6. Estructura de datos

- **Agenda de citas (Arreglo)**
  La agenda de citas fue implementada utilizando un arreglo, ya que este nos permite un acceso directo por índice y es una estructura adecuada para aplicar búsqueda binaria. Los datos fueron cargados desde archivos CSV dados por el docente, incluyendo conjuntos casi ordenados y desordenados. La agenda se ordena por el campo fechaHora que es un requisito previo para poder realizar búsquedas binarias.

- **Pacientes (Lista simplemente enlazada – SLL)**
  Para el manejo de pacientes se utilizó una lista simplemente enlazada (SLL) con los campos id, apellido y prioridad. Esta estructura nos resulta apropiada cuando no se requiere un acceso aleatorio y las operaciones principales consisten en recorridos de forma secuencial. Debido a la naturaleza de la estructura de los datos la búsqueda binaria no es apta, por lo que se implementaron búsquedas secuenciales.

- **Inventario (Arreglo)**
  El inventario se implementó haciendo uso de un arreglo, permitiendo ordenar los productos por el campo stock y realizar búsquedas binarias. De igual manera los datos fueron cargados desde el archivo CSV dado por el docente con el stock en orden inverso, lo que permitió analizar el comportamiento de los algoritmos de ordenación en un caso totalmente desfavorable. También se tuvo en cuenta la existencia de valores duplicados y casos bordes.

7. Discusión

¿Cuándo conviene usar búsqueda secuencial?

La búsqueda secuencial es recomendable cuando:

- La estructura de datos es pequeña.
- Los datos no están ordenados.
- Se trabaja con listas simplemente enlazadas (SLL), donde no existe acceso directo por índice.
- El costo de ordenar los datos supera el beneficio de aplicar búsqueda binaria.

¿Cuándo conviene usar búsqueda binaria?

La búsqueda binaria resulta más eficiente cuando:

- Se trabaja con arreglos ordenados.
- El volumen de datos es grande.
- Se realizan múltiples búsquedas sobre la misma estructura.

Uso de centinela y en el caso "no encontrado"

La técnica de centinela optimiza la búsqueda secuencial en arreglos al eliminar la verificación de límites dentro del bucle. Al insertar temporalmente la clave buscada como último elemento:

- Se garantiza la finalización del bucle.
- Se reduce el número de comparaciones.
- En el caso de "no encontrado", el algoritmo detecta esta situación al verificar si la coincidencia se produjo en la posición del centinela. Si ocurre allí, se concluye que el valor no estaba presente originalmente en el arreglo.
- Esta técnica es especialmente útil en arreglos grandes, donde pequeñas optimizaciones en el número de comparaciones pueden representar mejoras de rendimiento apreciables**.**

8. Tablas

Dataset	**n**	**Algoritmo**	**Comparaciones**	**Swaps**	**Tiempo (ns)**	**Conclusión**
Citas Ordenadas	100	Burbuja	4950	2309	370500	Comparaciones constantes
Citas Ordenadas	100	Selección	4950	94	196100	Comparaciones altas
Citas Ordenadas	100	Inserción	2403	2309	184900	Mejor Caso

Citas Casi Ordenadas	100	Burbuja	4797	341	97000	Escenario normal
Citas Casi Ordenadas	100	Selección	4950	5	108600	Peor Caso
Citas Casi Ordenadas	100	Inserción	440	341	54100	Mejor Caso
Inventario Inverso	500	Burbuja	124750	124750	6069700	Penalizado
Inventario Inverso	500	Selección	124750	250	3586000	Peor caso
Inventario Inverso	500	Inserción	124750	124750	4392100	Comportamiento o estable

```
>>> ANÁLISIS PARA AGENDA: citas_100.csv
Algoritmo    Comparaciones   Intercambios    Tiempo (ns)
-------------------------------------------------------------
Burbuja      4950            2309            370500
Seleccion    4950            94              196100
Insercion    2403            2309            184900

-------------------------------------------------------
```

```
--- PROCESANDO AGENDA: CASI ORDENADA ---
>> Analisis de algoritmos...

>>> ANÁLISIS PARA AGENDA: citas_100_casi_ordenadas.csv
Algoritmo    Comparaciones   Intercambios    Tiempo (ns)
-------------------------------------------------------------
Burbuja      4797            341             97000
Seleccion    4950            5               108600
Insercion    440             341             54100

-------------------------------------------------------
```

```
--- GESTION DE INVENTARIO ---
>> Cargando y ordenando por Stock...

>>> ANÁLISIS PARA INVENTARIO: inventario_500_inverso.csv
Algoritmo    Comparaciones   Intercambios    Tiempo (ns)
-------------------------------------------------------------
Burbuja      124750          124750          6069700
Seleccion    124750          250             3586000
Insercion    124750          124750          4392100

-------------------------------------------------------
```

9. Preguntas de Control:

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

  No es adecuada por la sencilla razón de que una búsqueda binaria se requiere saber tanto los valores anteriores como los posteriores, el problema de un Lista enlazada simple es que este solo conoce la dirección del siguiente valor, pero no se puede regresar, siendo esto necesario para la búsqueda binaria, además que se necesita el valor del medio en este caso se tendría que recorrer todos esos valores hasta llegar al punto y sin poder regresar, siendo una búsqueda incompatible para este tipo de estructura.

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

  Sin mucha ciencia este retorna el primero ya que lo que necesita es ir recorriendo arreglo, y si la clave del valor que queremos buscar es igual al que encuentra dentro del arreglo lo devuelve enseguida, deteniendo el recorrido.

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

  Se garantiza ya que como primer paso debe guardar el último elemento de nuestro arreglo original antes de colocar la key verificando primero que no se pierda la información, luego se restaura el valor original de la búsqueda después regresa a su estado original, además si ponemos en nuestro algoritmo también podremos distinguir si la coincidencia fue real o fue el centinela.

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

  En este caso como la binaria busca de acuerdo a la mitad, se utiliza modificaciones la cual sería el “lower bound” para que siga buscando a la izquierda hasta encontrar la primera coincidencia, y tambien está el “upper bound” para que siga buscando en este caso a la derecha hasta encontrar la última ocurrencia.

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

  Dos casos bordes más sencillos de detectar es el que el arreglo está vacío generando una inconsistencia al momento de usar algún método que reciba el arreglo, y la de que todos los elementos sean iguales, en el caso de la búsqueda binaria en este caso regresa cualquier índice pero ni la primera o la última, o incluso el upperBound puede generar algún tipo de bucle.

10. Resultados:

```
==================================
     DEMOSTRACIÓN COMPLETA UNL
==================================

----- BÚSQUEDA SECUENCIAL -----

Clave = 7
Array A: primera=-1, última=-1
Array B: primera=-1, última=-1
Array C: primera=-1, última=-1
Array D: primera=-1, última=-1

Clave = 5
Array A: primera=4, última=4
Array B: primera=0, última=0
Array C: primera=-1, última=-1
Array D: primera=-1, última=-1

Clave = 2
Array A: primera=1, última=1
Array B: primera=3, última=3
Array C: primera=-1, última=-1
Array D: primera=-1, última=-1

Clave = 42
Array A: primera=-1, última=-1
Array B: primera=-1, última=-1
Array C: primera=-1, última=-1
Array D: primera=-1, última=-1
```

```
----- BÚSQUEDA CON CENTINELA -----

Clave = 7
Array A: sentinelSearch=-1
Array B: sentinelSearch=-1
Array C: sentinelSearch=-1
Array D: sentinelSearch=-1

Clave = 5
Array A: sentinelSearch=4
Array B: sentinelSearch=0
Array C: sentinelSearch=-1
Array D: sentinelSearch=-1

Clave = 2
Array A: sentinelSearch=1
Array B: sentinelSearch=3
Array C: sentinelSearch=-1
Array D: sentinelSearch=-1

Clave = 42
Array A: sentinelSearch=-1
Array B: sentinelSearch=-1
Array C: sentinelSearch=-1
Array D: sentinelSearch=-1
```

```
----- BINARY SEARCH -----
Array A (ordenado): [1, 2, 3, 4, 5]
Key 7 -> binarySearch = -1
Key 5 -> binarySearch = 4
Key 2 -> binarySearch = 1
Key 42 -> binarySearch = -1

----- LOWERBOUND / UPPERBOUND -----
key=1 -> lowerBound=0, upperBound=1
key=2 -> lowerBound=1, upperBound=2
key=3 -> lowerBound=2, upperBound=3
key=4 -> lowerBound=3, upperBound=4
key=6 -> lowerBound=5, upperBound=5
key=10 -> lowerBound=5, upperBound=5

----- FINDALL EN ARRAY -----
Array A: [1, 2, 3, 4, 5]
Valores < 3 en A -> índices: [0, 1]

----- LISTA ENLAZADA (SLL) -----
Lista: 3 → 1 → 3 → 2
Primera ocurrencia de 3: 3
Última ocurrencia de 3 : 3
Nodos < 3: [1, 2]

----- INSERTION SORT -----
Original: [5, 4, 3, 2, 1]
Ordenado: [1, 2, 3, 4, 5]

=========== FIN DEMO ===========
```

Link del repositorio en GitHub:
https://github.com/RichardChambaS/Agenda_InventariosInteligentes.git

10. Conclusiones

- La elección del algoritmo depende del tamaño de la estructura y de si los datos están ordenados.
- El uso de predicados nos permite generalizar la búsqueda y reutilizar código.
- La técnica de centinela nos ayuda a reducir el número de comparaciones, optimizando la búsqueda secuencial en arreglos.