



## **Programación estructurada**

**Nombre:** María Celeste Carrasco Obando

**Docente:** José Durán

Lunes 24 de noviembre del 2025

## **Introducción**

Los algoritmos de búsqueda representan una de las herramientas fundamentales dentro de la ciencia de la computación, ya que permiten localizar información dentro de estructuras de datos de manera rápida y eficiente. En el presente caso de estudio, se desarrolla un prototipo básico de un sistema de búsqueda para una futura Biblioteca Digital Estudiantil.

El objetivo principal es comprender, aplicar y comparar diferentes algoritmos de búsqueda, tales como la búsqueda lineal, búsqueda binaria, búsqueda por coincidencias y la identificación de elementos máximos y mínimos. Además, se documenta el proceso académico siguiendo las normas APA 7, se desarrolla el código en C#, y se utiliza GitHub empleando buenas prácticas de control de versiones.

---

## **2. Marco Teórico**

### **2.1 ¿Qué es un algoritmo de búsqueda?**

Un algoritmo de búsqueda es un conjunto de pasos destinados a localizar un elemento específico dentro de una estructura de datos como listas, matrices, árboles o grafos. Estos algoritmos permiten encontrar información rápidamente, optimizar procesos y mejorar la eficiencia de los sistemas informáticos (Cormen et al., 2009).

---

### **2.2 Diferencias entre los tipos de búsqueda**

#### **Búsqueda lineal**

Recorre la estructura de datos elemento por elemento desde el inicio hasta el final.

- No requiere orden.
- Es simple pero puede ser lenta para listas grandes.
- Complejidad:  **$O(n)$** .

### **Búsqueda binaria**

Requiere que la lista esté ordenada. Se divide repetidamente en mitades para encontrar el valor buscado.

- Mucho más eficiente.
- Complejidad:  **$O(\log n)$** .

### **Búsqueda en estructuras no lineales**

Utiliza árboles, grafos u otras estructuras complejas.

Los algoritmos más usados incluyen:

- Búsqueda en anchura (BFS)
- Búsqueda en profundidad (DFS)

Muy utilizada en IA, rutas, motores de recomendación y videojuegos (Russell & Norvig, 2020).

---

### 2.3 Casos de uso reales

- **Sistemas operativos:** búsqueda de archivos y rutas (Microsoft, 2024).
  - **Motores de búsqueda:** análisis de palabras clave, indexación y ranking de resultados.
  - **Bases de datos:** utilización de índices para búsquedas rápidas (Oracle, 2024).
  - **IA y videojuegos:** caminos más cortos, toma de decisiones.
  - **Comercio electrónico:** filtros, coincidencias de texto, búsqueda por relevancia.
- 

### 2.4 Ejemplos en el mundo profesional

- **Google Maps:** algoritmos heurísticos como A\* para encontrar rutas óptimas.
- **Amazon:** búsqueda por coincidencias y ordenamiento por relevancia.
- **Bancos:** búsqueda binaria para localizar cuentas o transacciones ordenadas.

---

## 2.5 Ventajas y desventajas

Algoritmo	Ventajas	Desventajas
Búsqueda lineal	Fácil de implementar, funciona en listas no ordenadas	Ineficiente en listas grandes
Búsqueda binaria	Muy rápida y eficiente	Requiere orden previo
Búsqueda en árboles/grafos	Ideal para rutas y jerarquías	Más compleja de implementar

---

## 3. Explicación de los algoritmos implementados

### 3.1 Búsqueda lineal en lista de libros

Se recorre cada libro verificando si coincide con el título buscado.

Si se encuentra, se devuelve el libro; si no, se muestra un mensaje indicando que no existe.

Es ideal para este caso porque la lista no está ordenada.

---

### 3.2 Búsqueda binaria en lista ordenada de autores

Se ordena la lista alfabéticamente y luego se aplican divisiones consecutivas hasta hallar el autor.

Permite búsquedas más rápidas y eficientes.

---

### **3.3 Búsqueda del libro más reciente y más antiguo**

Se recorren todos los libros comparando los años de publicación.

Se mantienen dos variables:

- Libro con año máximo (más reciente).
  - Libro con año mínimo (más antiguo).
- 

### **3.4 Búsqueda de coincidencias en descripciones**

Se utiliza el método **Contains** para localizar palabras clave dentro de las descripciones de los libros.

Si no hay coincidencias, el sistema muestra un mensaje correspondiente, mejorando la experiencia del usuario.

---

## **4. Conclusiones**

El desarrollo de este caso de estudio permitió comprender la importancia de los algoritmos de búsqueda en la computación y en los sistemas modernos. Implementar estos algoritmos en C# fortaleció la lógica de programación y el uso de estructuras de datos. Además, trabajar con GitHub facilitó el control de versiones, la organización y el flujo profesional de desarrollo de software.

Se concluye que la búsqueda eficiente es esencial para sistemas reales como bases de datos, motores de búsqueda e inteligencia artificial. Un algoritmo ineficiente puede generar tiempos de respuesta lentos, fallas de rendimiento y mala experiencia del usuario.

## **Reflexión individual**

La búsqueda es un concepto fundamental en la ciencia de la computación y en el desarrollo de software porque permite localizar información de manera rápida y eficiente dentro de un sistema. En estructuras de datos, la búsqueda es esencial para aprovechar al máximo la organización de la información; por ejemplo, una lista ordenada, un árbol o una tabla hash permiten encontrar elementos mucho más rápido que una estructura mal organizada. Sin buenas técnicas de búsqueda, estas estructuras perderían gran parte de su utilidad.

En los sistemas reales, la búsqueda tiene un impacto enorme. Las bases de datos dependen de algoritmos de búsqueda para responder consultas en fracciones de segundo. Los motores de búsqueda como Google necesitan técnicas especializadas para revisar millones de páginas web. Incluso la inteligencia artificial usa procesos de búsqueda para tomar decisiones,

encontrar rutas óptimas o analizar grandes cantidades de datos. Sin algoritmos de búsqueda eficientes, muchos de estos sistemas simplemente no funcionarían.

Cuando un algoritmo de búsqueda es ineficiente, las consecuencias pueden ser graves: los programas se vuelven lentos, consumen más recursos y pueden fallar cuando manejan grandes volúmenes de datos. En aplicaciones críticas, como sistemas bancarios o plataformas que procesan miles de solicitudes por segundo, un algoritmo lento puede generar retrasos importantes o incluso colapsar el sistema. Por eso, entender y elegir bien el tipo de búsqueda es clave para desarrollar software de calidad.

## Referencias

Adogy. (2025). *Algoritmo de búsqueda.*

<https://www.adogy.com/es/t%C3%A9rminos/algoritmo-de-b%C3%BAsqueda/adogy.com>

GeeksforGeeks. (s. f.). *Linear Search Algorithm.*

<https://www.geeksforgeeks.org/linear-search/> GeeksforGeeks

GeeksforGeeks. (s. f.). *Linear Search vs Binary Search.*

<https://www.geeksforgeeks.org/dsa/linear-search-vs-binary-search/> GeeksforGeeks

Make It Real. (s. f.). *Algoritmos de búsqueda.*

<https://guias.makeitreal.camp/docs/algoritmos/busqueda> guias.makeitreal.camp

UN Salesiana de Bolivia. (s. f.). *Algoritmo de Búsqueda Binaria* [Documento PDF].

<https://virtual.usalesiana.edu.bo/web/conte/archivos/149.pdf> virtual.usalesiana.edu.bo

CPC-Gallos. (2024). *Algoritmos de búsqueda* [Presentación PDF].

[https://cpc-gallos.github.io/Presentations/2024/2\\_005-Algoritmos\\_Busqueda.pdf](https://cpc-gallos.github.io/Presentations/2024/2_005-Algoritmos_Busqueda.pdf) CPC GALLOS

Wiki OIA UNSAM. (s. f.). *algoritmos-oia:busqueda-binaria.*

<https://wiki.oia.unsam.edu.ar/algoritmos-oia/busqueda-binaria> wiki.oia.unsam.edu.ar

Wikipedia. (s. f.). *Binary search tree.*

[https://en.wikipedia.org/wiki/Binary\\_search\\_tree](https://en.wikipedia.org/wiki/Binary_search_tree) Wikipedia

GeeksforGeeks. (2024). *Searching Algorithms in Data Structures.*

<https://www.geeksforgeeks.org/searching-algorithms>

IBM. (2024). *What is a Search Algorithm?*

<https://www.ibm.com/topics/search-algorithm>

Microsoft Learn. (2024). *Data Structures and Algorithms*.

<https://learn.microsoft.com/en-us/shows/data-structures-algorithms>


Oracle. (2024). *Database Indexing and Searching*. <https://docs.oracle.com>

## **Anexos**

## Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).  
Required fields are marked with an asterisk (\*).

- General**

**Owner \***  Celeste-10 / **Repository name \***

✔ Your new repository will be created as **Caso-de-estudio**.  
The repository name can only contain ASCII letters, digits, and the characters -, ., and \_.

Great repository names are short and memorable. How about **symmetrical-pancake**?

**Description**


0 / 350 characters
- Configuration**

**Choose visibility \***  
Choose who can see and commit to this repository Public


**Add README** Off  
READMEs can be used as longer descriptions. [About READMEs](#)




**Add .gitignore** No .gitignore  
.gitignore tells git which files not to track. [About ignoring files](#)

**Add license** No license

 **Caso-de-estudio** Public Pin Watch 0

main 4 Branches 0 Tags  t Add file <> Code

 **Celeste-10** Format README.md for better readability f4f0426 · now 5 Commits

 Caso de estudio1	Agregar funcion binaria	13 hours ago
 Caso de estudio1.sln	Agregar funcion binaria	13 hours ago
 README.md	Format README.md for better readability	now

## Repositorio

<https://github.com/Celeste-10/Caso-de-estudio.git>