Universidad Americana (UAM)

Informe Final: *Sistemas de búsqueda para una biblioteca digital estudiantil*
Asignatura: *Programación estructurada | Grupo 1*
Docente: *Ms José Duran García*

Estudiantes: Daniella Rocha y Franklin Callejas
Fecha: 24 de noviembre de 2025

1. INTRODUCCIÓN

El presente informe técnico detalla el proceso de investigación, diseño y desarrollo de un prototipo de software para la futura Biblioteca Digital Estudiantil de la universidad. En el contexto actual de la gestión de la información, la capacidad de recuperar datos de manera eficiente es crítica para la experiencia del usuario y el rendimiento del sistema.

El objetivo principal de este caso de estudio fue la implementación manual de algoritmos fundamentales de búsqueda (lineal y binaria) y estructuras de datos en el lenguaje C#, evitando el uso de funciones predefinidas de alto nivel para comprender la lógica subyacente de estos procesos. Asimismo, se integraron prácticas de desarrollo colaborativo mediante el uso de control de versiones con Git y GitHub.

2. MARCO TEÓRICO

2.1. Algoritmos de Búsqueda

Un algoritmo de búsqueda es un procedimiento computacional diseñado para localizar un elemento específico, denominado "clave" o "target", dentro de una colección de datos. Su función es determinar la existencia de dicho elemento y retornar su ubicación o el objeto asociado (Joyanes Aguilar, 2008).

2.2. Búsqueda Lineal vs. Búsqueda Binaria

La **Búsqueda Lineal** es el método más sencillo, el cual recorre la estructura secuencialmente de principio a fin. Aunque es fácil de implementar y no requiere que los datos estén ordenados, su eficiencia es baja en grandes volúmenes de datos, con una complejidad temporal de $O(n)$ en el peor caso.

Por otro lado, la **Búsqueda Binaria** utiliza la estrategia de "divide y vencerás". Divide repetidamente el conjunto de datos en dos mitades, descartando aquella donde no puede estar el dato buscado. Este método es altamente eficiente, con una complejidad de $O(\log n)$, pero tiene como requisito obligatorio que la lista esté ordenada previamente (Cormen et al., 2009).

2.3. Búsqueda en Estructuras No Lineales y Texto

En escenarios más complejos, como la búsqueda de patrones dentro de texto (cadenas de caracteres), se requieren algoritmos que comparen secuencias carácter por carácter. En el ámbito profesional, motores de búsqueda y bases de datos utilizan variantes avanzadas de estos algoritmos, como árboles B o índices invertidos, para gestionar millones de registros.

3. EXPLICACIÓN DE ALGORITMOS IMPLEMENTADOS

Para cumplir con los requerimientos del caso de estudio, se desarrolló una aplicación en **Windows Forms (.NET)** que gestiona una colección de objetos `Libro`. A continuación, se detallan las implementaciones técnicas:

A. Módulo de Búsqueda Lineal (Por Título)

Se implementó un ciclo iterativo (foreach) que recorre la lista completa de libros. En cada iteración, se compara el título del libro actual con el valor ingresado por el usuario. Se utilizó una conversión a minúsculas (ToLower) para normalizar la entrada, pero la comparación lógica se realizó de manera exacta para garantizar la precisión del resultado.

B. Módulo de Búsqueda Binaria (Por Autor)

Dado que la búsqueda binaria requiere datos ordenados, se implementó previamente un algoritmo de **Ordenamiento Burbuja (Bubble Sort)** manual. Este reorganiza la lista de libros alfabéticamente según el campo "Autor" antes de iniciar la búsqueda. Posteriormente, el algoritmo binario calcula el índice central y decide si buscar en la mitad izquierda o derecha comparando las cadenas de texto.

C. Reporte de Extremos

Para encontrar el libro más antiguo y el más reciente, se utilizó un recorrido lineal simple (O(n)). Se inicializaron dos variables de referencia con el primer elemento de la lista y se comparó el campo Anio de cada libro subsiguiente, actualizando las referencias cuando se encontraba un valor menor (antiguo) o mayor (reciente).

D. Búsqueda en Descripción (Coincidencia de Patrones)

Se desarrolló un algoritmo de búsqueda de texto manual. Este utiliza dos bucles anidados: el externo recorre el texto de la descripción y el interno compara la secuencia de caracteres con la palabra clave buscada. Esto permite detectar coincidencias parciales sin utilizar métodos prefabricados como Contains().

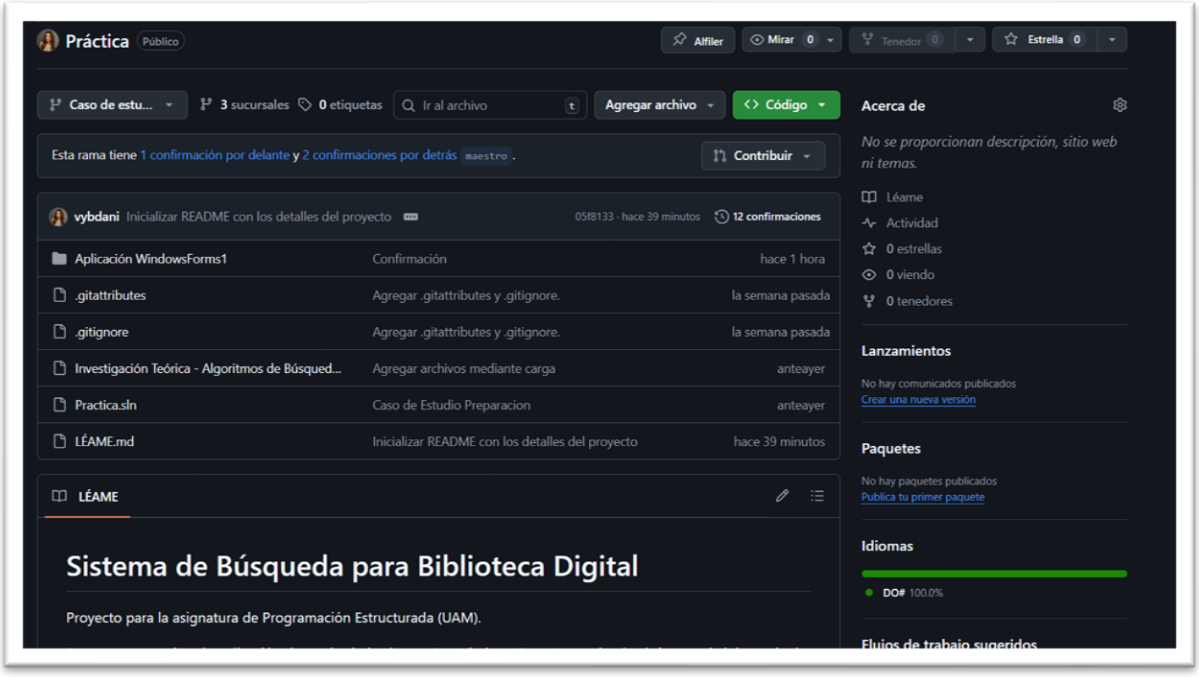
4. EVIDENCIA DEL REPOSITORIO Y FUNCIONAMIENTO

4.1. Repositorio en GitHub

El proyecto fue gestionado mediante Git, utilizando la rama (Caso-De-Estudio)

URL del Repositorio: ***https://github.com/vybdani/Practica.git***

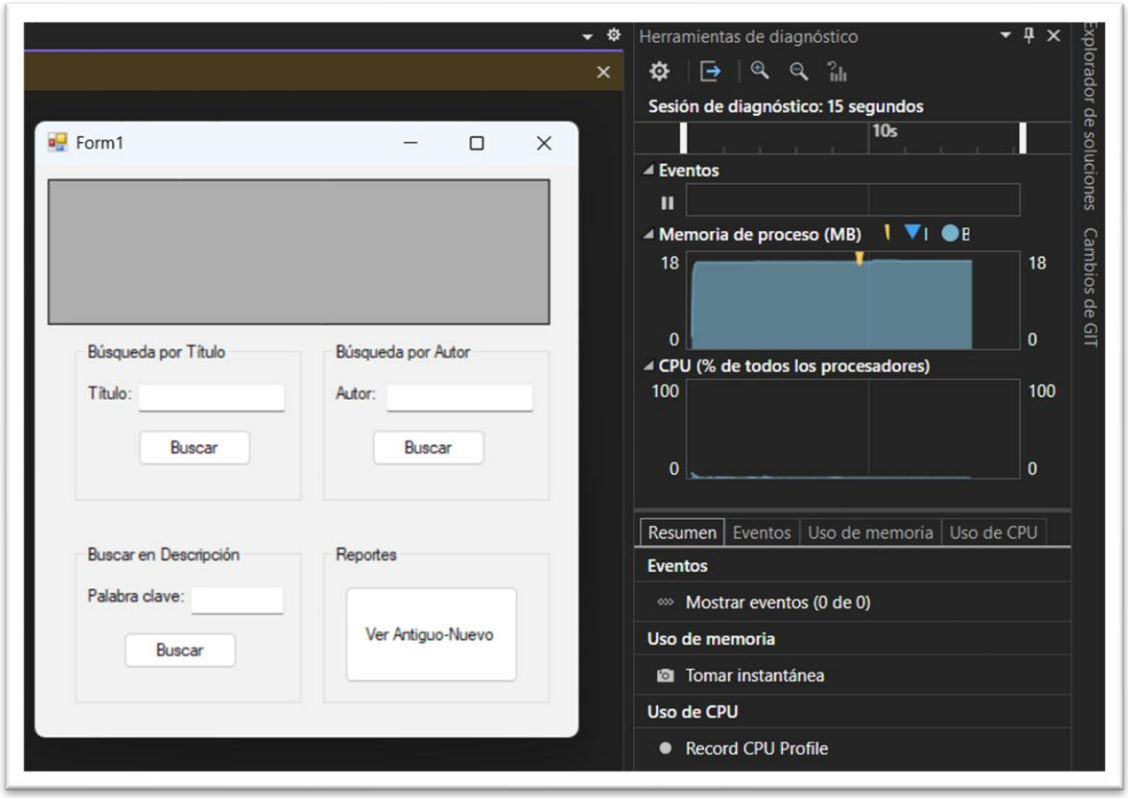
Captura 1: Estructura del Repositorio y README



4.2. Ejecución del Software

A continuación, se muestra la interfaz gráfica desarrollada en Visual Studio Community 2022.

Captura 2: Interfaz de Usuario



5. CONCLUSIONES

La realización de este proyecto permitió evidenciar la importancia crítica de elegir el algoritmo adecuado según el contexto de los datos. Se observó que, si bien la búsqueda lineal es eficaz para listas pequeñas y desordenadas, la búsqueda binaria ofrece un rendimiento superior, aunque conlleva el "costo" computacional de mantener los datos ordenados.

Asimismo, la implementación manual de estos algoritmos reforzó la comprensión lógica de las estructuras de control (bucles y condicionales) y el manejo de objetos en memoria. Finalmente, el uso de GitHub facilitó la organización del código y simuló un flujo de trabajo colaborativo real, una competencia esencial en el desarrollo de software profesional.

6. REFERENCIAS BIBLIOGRÁFICAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3a ed.). MIT Press.

Joyanes Aguilar, L. (2008). Fundamentos de programación: Algoritmos, estructura de datos y objetos (4a ed.). McGraw-Hill Interamericana.

Microsoft. (2023). Documentación de C\# y .NET. Microsoft Learn.