# Trabajo Integrador: Programación

**Materia:** Programación
**Profesor/a:** Ariel Enferrel
**Alumnos/as:** Chiappone Michael
Campana Jonatan
**Fecha de entrega**: 09/06/2025

# Contenido

## Introducción

El presente trabajo tiene como objetivo abordar el estudio e implementación de algoritmos de búsqueda utilizando el lenguaje de programación Python. Se trata de un tema fundamental dentro del campo de la programación, ya que permite resolver problemas comunes como la localización de datos dentro de estructuras, optimizando tiempo y recursos computacionales.

La elección de este tema se fundamenta en su aplicabilidad práctica y su presencia en múltiples áreas de desarrollo de software, desde la manipulación de listas simples hasta la gestión de grandes volúmenes de información. A través del desarrollo de un programa interactivo en consola, se pretende comprender, comparar e implementar dos algoritmos de búsqueda clásicos: la búsqueda lineal y la búsqueda binaria.

El trabajo busca fortalecer el razonamiento lógico, mejorar las habilidades en el uso de Python, y promover el análisis crítico en torno a la eficiencia de los algoritmos. Además, se integra la documentación del proceso y la presentación de resultados, cumpliendo con los criterios de la cátedra de Programación I.

## Marco Teórico

Los algoritmos de búsqueda son técnicas utilizadas para encontrar la posición de un elemento dentro de una estructura de datos, generalmente una lista o arreglo. Existen distintos tipos de algoritmos de búsqueda, cada uno con su propia lógica, ventajas y desventajas.

### Búsqueda Lineal

La búsqueda lineal, también conocida como búsqueda secuencial, consiste en recorrer uno por uno los elementos de la lista hasta encontrar el valor buscado. Es simple de implementar y no requiere que la lista esté ordenada.

Ventajas:

Fácil implementación.

Funciona en listas no ordenadas.

Desventajas:

Poco eficiente para listas grandes.

Complejidad Temporal:

Mejor caso: O(1) → el valor está al principio.

Peor caso: O(n) → el valor no está o está al final.

### Búsqueda Binaria

La búsqueda binaria es más eficiente, pero requiere que la lista esté ordenada previamente. Su funcionamiento se basa en dividir el espacio de búsqueda a la mitad repetidamente, descartando la mitad donde no puede estar el elemento.

Ventajas:

Muy rápida para listas grandes ordenadas.

Reduce el número de comparaciones.

Desventajas:

Requiere listas ordenadas.

Complejidad Temporal:

Mejor caso: O(1)

Peor caso: $O(\log_2 n)$

| Comparación general | | | |
|---|---|---|---|
| Algoritmo | Estructura Requerida | Eficiencia (Peor Caso) | Ventaja principal |
| Búsqueda lineal | Lista no ordenada | O(n) | Simple y sin requisitos previos |
| Búsqueda binaria | Lista ordenada | O(log n) | Rápida en listas grandes |

**Caso Práctico**

Se desarrolló un script en Python que permite al usuario:

Crear una lista personalizada o usar una lista predefinida.
Elegir entre búsqueda lineal o binaria.
Ingresar un número a buscar.
Recibir una respuesta clara indicando si el número fue encontrado y en qué índice.
El código se encuentra modularizado y documentado. Las funciones principales son:

busqueda_lineal(valor, lista)
busqueda_binaria(valor, lista)
crear_lista(), seleccionar_lista()

Menú interactivo para elegir opciones.
Capturas del programa funcionando se incluyen en los anexos.

## Metodología Utilizada

El trabajo fue abordado mediante los siguientes pasos:

Selección del tema: Se eligieron algoritmos de búsqueda por su relevancia básica en programación.

Investigación previa: Se consultaron fuentes oficiales de Python y sitios especializados.

Diseño del algoritmo: Se decidió implementar búsqueda lineal y binaria, con menú interactivo.

Desarrollo del código: Se utilizó Python 3 y se trabajó con funciones para modularidad.

Pruebas: Se realizaron búsquedas con listas ordenadas y desordenadas, usando diferentes valores.

Documentación: Se redactó el código con comentarios y se preparó un informe.

Repositorio Git: El proyecto fue subido a GitHub.

Producción del video: Se realizó una grabación explicando el proyecto, su lógica y reflexiones finales.

## Resultados Obtenidos

Durante las pruebas del programa se obtuvieron los siguientes resultados:

Búsqueda lineal encontró correctamente los valores, pero tardó más en listas largas.

Búsqueda binaria fue más eficiente, pero requería ordenar previamente la lista.

Se comprobó la diferencia de eficiencia con casos extremos (elemento al inicio, al final y ausente).

El menú fue intuitivo y no presentó errores de ejecución.

El código funcionó correctamente tanto en IDEs como en terminal.

Se validaron los resultados con listas de prueba, mostrando que los algoritmos se comportan como se espera según la teoría.

## Conclusiones

Este trabajo permitió comprender e implementar dos algoritmos fundamentales en programación: la búsqueda lineal y la búsqueda binaria. Se fortalecieron habilidades de codificación, diseño lógico y validación de resultados.
Se comprobó empíricamente la mayor eficiencia de la búsqueda binaria frente a la lineal, pero también su limitación de requerir listas ordenadas. Esta experiencia refuerza la importancia de elegir el algoritmo adecuado según el contexto del problema.

Además, el trabajo promovió el trabajo colaborativo, el uso de buenas prácticas de programación, y la comunicación efectiva mediante un video explicativo.
Como mejora futura se podrían incorporar algoritmos de ordenamiento para complementar el proyecto y analizar cómo impacta la preparación de la lista sobre el rendimiento general.

## Bibliografía

Python Software Foundation. (2024). Python 3 Documentation. https://docs.python.org/3/

GeeksForGeeks. (2024). Python Search Algorithms. https://www.geeksforgeeks.org/

W3Schools. (2024). Python Search Methods. https://www.w3schools.com/python/

Sweigart, A. (2019). Automate the Boring Stuff with Python. No Starch Press.

Material de la cátedra de Programación I – Universidad Tecnológica Nacional.

## Anexos

Capturas de pantalla del programa funcionando (búsqueda lineal y binaria).

Código fuente comentado (busqueda.py).

Archivo README del proyecto.

Enlace al video explicativo:

https://youtu.be/FOPMoAV6sf4

Enlace al repositorio GitHub:

https://github.com/CampanaJ/trabajo_integrador_informatica