

# TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN A DISTANCIA

## Trabajo Integrador: Algoritmos de Búsqueda y Ordenamiento en Python

**Alumno:**

Gomez Elias Walter, [gomezelias1790@gmail.com](mailto:gomezelias1790@gmail.com)

Frasconá Francisco, [franciscofrascona@gmail.com](mailto:franciscofrascona@gmail.com)

**Materia:** Programación I

**Profesor/a:** Julieta Trapé

**Fecha de entrega:** 09/06/2025

---

### Índice

1. Introducción
  2. Marco Teórico
  3. Caso Práctico
  4. Metodología Utilizada
  5. Resultados Obtenidos
  6. Conclusiones
  7. Bibliografía
  8. Anexos
- 

### 1. Introducción

El tratamiento eficiente de grandes volúmenes de datos es una necesidad cada vez más importante en el ámbito de la informática. Dentro de las herramientas más utilizadas para este fin se encuentran los algoritmos de búsqueda y ordenamiento. Este trabajo se enfoca en la implementación de dos algoritmos fundamentales: el algoritmo de ordenamiento Bubble Sort y el algoritmo de búsqueda Búsqueda Binaria. Ambos se aplicarán en un lenguaje de programación de alto nivel como Python, con el objetivo de fortalecer el razonamiento lógico y la comprensión de estructuras algorítmicas fundamentales.

El propósito de este trabajo no solo es implementar estos algoritmos, sino también evaluar su rendimiento, comprender su lógica interna, analizar su complejidad computacional y reconocer en qué contextos resultan útiles. Además, se busca consolidar los conocimientos adquiridos durante la cursada de Programación I, aplicando herramientas básicas como listas, funciones y estructuras condicionales en la resolución de un problema práctico.

---

### 2. Marco Teórico

#### 2.1 Algoritmos de Ordenamiento

Un algoritmo de ordenamiento organiza un conjunto de datos (por lo general una lista o un arreglo) en un orden específico, ya sea ascendente o descendente. La eficiencia de un algoritmo se mide principalmente por su **complejidad temporal**, es decir, el tiempo que tarda en ejecutarse según la cantidad de datos.

### Bubble Sort

Bubble Sort (Ordenamiento Burbuja) es uno de los algoritmos más simples de ordenamiento. Funciona comparando pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que la lista está ordenada. Su principal ventaja es la simplicidad, aunque su eficiencia es limitada para grandes volúmenes de datos.

- Complejidad temporal:  $O(n^2)$  en el peor y promedio de los casos.
- Complejidad espacial:  $O(1)$ , ya que es un algoritmo in-place (no necesita estructuras auxiliares).
- Ventaja: fácil de implementar y entender.
- Desventaja: poco eficiente comparado con otros algoritmos como MergeSort o QuickSort.

### 2.2 Algoritmos de Búsqueda

Un algoritmo de búsqueda localiza la posición de un elemento dentro de una estructura de datos. Entre los algoritmos más utilizados se encuentra la Búsqueda Binaria, que mejora considerablemente el tiempo de búsqueda en comparación con una búsqueda lineal, pero requiere que los datos estén previamente ordenados.

#### Búsqueda Binaria

La Búsqueda Binaria trabaja dividiendo el conjunto en mitades sucesivas, comparando el elemento medio con el objetivo. Si no hay coincidencia, se determina en qué mitad continuar la búsqueda. Este enfoque reduce significativamente el número de comparaciones necesarias.

- Requisito: los datos deben estar ordenados.
- Complejidad temporal:  $O(\log n)$ .
- Complejidad espacial:  $O(1)$ .
- Ventaja: muy eficiente para conjuntos grandes.
- Desventaja: requiere listas ordenadas.

### 2.3 Aplicaciones prácticas

- Búsqueda de productos en catálogos digitales.
- Organización alfabética o numérica de registros.
- Indexación en bases de datos.
- Implementación de funcionalidades básicas en software.

#### Búsqueda Lineal

La búsqueda lineal, también llamada búsqueda secuencial, es el algoritmo más simple para localizar un elemento en una lista. Consiste en recorrer uno a uno todos los elementos del conjunto hasta encontrar el valor buscado o llegar al final.

- Requisito: no es necesario que la lista esté ordenada.
  - Complejidad temporal:  $O(n)$ .
  - Complejidad espacial:  $O(1)$ .
  - Ventajas: fácil de implementar; útil en listas pequeñas o no ordenadas.
  - Desventajas: poco eficiente para listas grandes debido a su crecimiento lineal en tiempo de ejecución.
- 

### 3. Caso Práctico

#### Descripción

Se desarrolló un programa en Python que simula una biblioteca digital interactiva, permitiendo al usuario gestionar una colección de libros a través de un menú simple, utilizando el algoritmo Bubble Sort. Luego, sobre esa lista ya ordenada, se aplica el algoritmo de búsqueda binaria para localizar en la lista un título ingresado por el usuario. La implementación se realiza utilizando estructuras básicas del lenguaje Python, sin emplear librerías externas.

El programa presenta las siguientes funcionalidades:

- Mostrar la lista de libros disponibles.
- Ordenar los títulos de forma ascendente o descendente utilizando el algoritmo Bubble Sort.
- Buscar un título específico mediante el algoritmo de Búsqueda Binaria, que requiere que la lista esté previamente ordenada. Si el usuario intenta buscar sin haber ordenado, el sistema lo hace automáticamente en orden ascendente.

El desarrollo se estructura de la siguiente manera:

- mostrar\_libros(): imprime en pantalla la lista actual de libros.
- bubble\_sort(): ordena los libros comparando títulos y aplicando el método Bubble Sort, configurable en orden ascendente o descendente.
- busqueda\_binaria(): permite encontrar un título específico dentro de la lista ya ordenada.
- menu(): gestiona la interacción del usuario con el programa mediante un menú en consola.

#### Código Fuente

```
# Simulación de una Biblioteca - Búsqueda y Ordenamiento con Bubble Sort,
Búsqueda Binaria y Lineal

def mostrar_libros(libros):
    print("\n■ Lista de libros disponibles:")
    for i, libro in enumerate(libros):
        print(f"{i + 1}. {libro}")

def bubble_sort(libros, ascendente=True):
    n = len(libros)
    for i in range(n):
        for j in range(0, n-i-1):
            if (ascendente and libros[j].lower() > libros[j+1].lower()) or (not ascendente and libros[j].lower() < libros[j+1].lower()):
                libros[j], libros[j+1] = libros[j+1], libros[j]
    return libros

def busqueda_binaria(libros, titulo):
    low = 0
    high = len(libros) - 1
    while low <= high:
        mid = (low + high) // 2
        if libros[mid].lower() == titulo.lower():
            return True
        elif libros[mid].lower() < titulo.lower():
            low = mid + 1
        else:
            high = mid - 1
    return False

def busqueda_lineal(libros, titulo):
    for libro in libros:
        if libro.lower() == titulo.lower():
            return True
    return False

def menu():
    libros = [
        "El Principito",
        "Cien Años de Soledad",
        "Rayuela",
        "Don Quijote",
        "Ficciones",
        "La Odisea",
        "1984",
```

```
"Crimen y Castigo"
]

while True:
    print("\n ◇ Menú Biblioteca ◇ ")
    print("1. Mostrar lista de libros")
    print("2. Ordenar libros (Bubble Sort)")
    print("3. Buscar un libro")
    print("4. Salir")

    opcion = input("Elige una opción: ")

    if opcion == "1":
        mostrar_libros(libros)

    elif opcion == "2":
        orden = input("¿Orden ascendente (A) o descendente (D)?").lower()
        if orden == "a":
            libros = bubble_sort(libros, ascendente=True)
            print("✓ Libros ordenados ascendente (Bubble Sort).")
        elif orden == "d":
            libros = bubble_sort(libros, ascendente=False)
            print("✓ Libros ordenados descendente (Bubble Sort).")
        else:
            print("✗ Opción inválida.")

    elif opcion == "3":
        metodo = input("¿Usar búsqueda binaria (B) o lineal (L)?").lower()
        titulo = input("🔍 Ingresá el título del libro a buscar: ")

        if metodo == "b":
            if libros != sorted(libros, key=lambda x: x.lower()):
                print("⚠ Los libros deben estar ordenados ascendente para usar búsqueda binaria. Ordenándolos automáticamente...")
                libros = bubble_sort(libros, ascendente=True)
            encontrado = busqueda_binaria(libros, titulo)
        elif metodo == "l":
            encontrado = busqueda_lineal(libros, titulo)
        else:
            print("✗ Método de búsqueda inválido.")
            continue

    elif opcion == "4":
        print("Hasta luego!")
        break
```

```
if encontrado:
    print(f"✅ El libro '{titulo}' está en la biblioteca.")
else:
    print(f"❌ El libro '{titulo}' no se encuentra.")

elif opcion == "4":
    print("👋 Saliendo del sistema. ¡Hasta luego!")
    break

else:
    print("⚠️ Opción no válida. Intenta de nuevo.")

# Ejecutar el programa
menu()
```

#### 4. Metodología Utilizada

1. Investigación teórica en fuentes académicas y documentación oficial de Python.
  2. Desarrollo del código desde cero con estructuras propias del lenguaje.
  3. Validación del comportamiento de los algoritmos con distintas entradas.
  4. Comparación de tiempos de ejecución con listas más grandes para analizar eficiencia.
  5. Elaboración del informe y planificación del video explicativo.
- 

#### 5. Resultados Obtenidos

- El algoritmo Bubble Sort logró ordenar listas de distintos tamaños correctamente.
  - La búsqueda binaria encontró exitosamente elementos cuando estaban presentes en la lista ordenada.
  - Se comprobó que la búsqueda binaria no funciona si los datos no están previamente ordenados.
  - El código se ejecutó correctamente en el entorno de desarrollo local sin errores de compilación ni ejecución.
  - Se comprobó que el uso de algoritmos eficientes mejora considerablemente el rendimiento en listas más grandes.
- 

#### 6. Conclusiones

La implementación de estos algoritmos permitió reforzar los fundamentos de la programación estructurada, especialmente en el uso de bucles, condicionales y manejo de listas. También se entendió la diferencia entre algoritmos eficientes y aquellos que, aunque simples, no son adecuados para ciertos contextos. Como mejora futura, se propone ampliar el proyecto con visualizaciones gráficas que permitan observar el comportamiento paso a paso de los algoritmos o integrar la comparación entre distintos métodos de ordenamiento.

---

## 7. Bibliografía

- Python Software Foundation. (2024). Python 3 Documentation.  
<https://docs.python.org/3/>
  - Khan Academy: <https://es.khanacademy.org/computing/computer-science/algorithms>
  - Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
  - GeeksForGeeks. (2024). Data Structures and Algorithms.  
<https://www.geeksforgeeks.org>
- 

## 8. Anexos

- Capturas de pantalla del código ejecutado.
- Enlace al video explicativo: [insertar link].
- Repositorio de código en GitHub: [insertar link].