

**Alumnos:**

**Zarate Gaston** - Gastonzarate25@gmail.com

**Romero Alani Elvio Nahuel** - [nahuelromero43@outlook.es](mailto:nahuelromero43@outlook.es)

**Materia:** Sistemas Operativos

**Profesor:** Claudia Rigoni - Comisión 5

**Fecha de Entrega:** 9 de junio de 2025

---

## 1. Introducción

En el contexto actual, la gestión eficiente de la información resulta fundamental para la toma de decisiones dentro de las organizaciones. Entre las tareas clave de los sistemas de información se encuentran la búsqueda y el ordenamiento de datos, procesos que permiten acceder rápidamente a información relevante y optimizar recursos. En este trabajo, se abordan diferentes algoritmos de búsqueda y ordenamiento desde una perspectiva práctica, mediante el desarrollo de un sistema simulado de inventario para una tienda. En dicho sistema, se aplican distintos algoritmos para localizar productos y organizarlos según criterios como el precio o el stock disponible.

---

## 2. Marco Teórico

### Búsqueda

Un algoritmo de búsqueda tiene como propósito encontrar un dato específico dentro de una estructura, como una lista o un arreglo.

- Búsqueda lineal:

Recorre secuencialmente cada elemento hasta encontrar el deseado o llegar al final.

Es simple de implementar y útil para listas pequeñas o sin orden, pero su complejidad es  $O(n)$ , lo que la vuelve ineficiente a medida que crece la cantidad de datos.

- **Búsqueda binaria:**

Requiere que los datos estén ordenados previamente. Divide el espacio de búsqueda por la mitad en cada iteración, reduciendo drásticamente la cantidad de comparaciones necesarias. Tiene una complejidad de  $O(\log n)$ , lo que la hace mucho más rápida para listas grandes.

- **Otros tipos de búsqueda (opcional para citar):**

También existen algoritmos como la búsqueda por interpolación (eficaz en listas con distribución uniforme) y la búsqueda hash, que permite acceder a datos en tiempo constante  $O(1)$  mediante funciones hash.

La eficiencia de la búsqueda depende directamente del tamaño de los datos y de si están ordenados o no. Según la tabla vista en el material, mientras que una búsqueda lineal sobre 100,000 elementos puede requerir 100,000 comparaciones, la búsqueda binaria puede resolverla en apenas 16 pasos.

---

## Ordenamiento

Ordenar datos implica organizarlos según un criterio definido, como precio, stock, nombre, o fecha. El ordenamiento no solo mejora la legibilidad, sino que facilita otras operaciones, como búsquedas más eficientes o comparaciones.

Entre los algoritmos de ordenamiento más conocidos se encuentran:

- **Bubble Sort:**

Compara elementos adyacentes y los intercambia si están en el orden incorrecto. Su

complejidad es  $O(n^2)$ , lo que lo hace ineficiente en listas grandes, aunque su lógica es simple y útil con fines didácticos.

- Quick Sort:

Utiliza la estrategia de *divide y vencerás*, eligiendo un "pivote" y reordenando los elementos alrededor de él, recurriendo luego sobre las sublistas. Tiene una eficiencia promedio de  $O(n \log n)$  y suele ser más rápido que otros métodos en la práctica.

- Otros algoritmos comunes:

- Selection Sort: encuentra el menor elemento y lo coloca al inicio.
- Insertion Sort: inserta cada nuevo elemento en su posición correcta dentro de una sublista ya ordenada.
- Merge Sort: divide la lista y la ordena por mitades, fusionándolas después.  
Muy eficiente y estable.

La elección del algoritmo adecuado dependerá de factores como el tamaño del conjunto de datos, su grado de desorden, el tiempo disponible y la memoria necesaria (complejidad espacial).

---

## **Análisis de eficiencia y complejidad**

El análisis de algoritmos permite comparar su rendimiento usando la notación Big-O, que describe el crecimiento del tiempo o la memoria en función del tamaño de entrada ( $n$ ):

Complejidad	Nombre	Ejemplo común
$O(1)$	Constante	Acceso a índice en array
$O(\log n)$	Logarítmica	Búsqueda binaria
$O(n)$	Lineal	Búsqueda lineal
$O(n \log n)$	Cuasilineal	Quick sort, Merge sort
$O(n^2)$	Cuadrática	Bubble sort, Selection sort

Además de la complejidad teórica, se puede hacer análisis empírico mediante pruebas reales, midiendo el tiempo de ejecución con herramientas como time o timeit en Python.

### Importancia práctica

Estos algoritmos son claves en:

- Bases de datos: búsquedas rápidas sobre grandes volúmenes.
- E-commerce: ordenar productos por precio, relevancia o stock.
- IA y optimización: encontrar soluciones óptimas entre miles de posibilidades.
- Procesamiento de datos: limpieza, clasificación y análisis.

En resumen, los algoritmos de búsqueda y ordenamiento mejoran la eficiencia, organización y escalabilidad de los sistemas informáticos, siendo una herramienta esencial para desarrolladores y analistas.

---

### 3. Caso Práctico

Simulación de sistema de inventario:

Para aplicar y observar el funcionamiento de los algoritmos mencionados, se desarrolló un sistema en python que simula la gestión de inventario de una tienda. Cada producto del inventario posee atributos como ID, Nombre, Stock y Precio.

El sistema implementa

- Implementación de archivo .csv con una lista predefinida de productos con ID, precio y stock.
- Búsqueda lineal y binaria por nombre de producto.
- Ordenamiento por precio utilizando el algoritmo Bubble Sort(Creando o sobrescribiendo archivo .csv con productos ordenados).
- Ordenamiento por stock utilizando el algoritmo Quick Sort(Creando o sobrescribiendo archivo .csv con productos ordenados).

Esta simulación permite visualizar en la práctica cómo se comportan los algoritmos en distintos contextos, tal como ocurriría en un sistema real de punto de venta, administración de stock o control de inventario.

---

#### 4. Metodología Utilizada

1. Selección del tema: Se optó por búsqueda y ordenamiento por ser conceptos base y de gran aplicabilidad.
  2. Investigación teórica: Se consultaron fuentes académicas y documentación oficial de Python.
  3. Diseño del sistema: Se definió una clase Producto y una lista de productos aleatorios.
  4. Implementación: Se programaron los algoritmos y se aplicaron sobre el inventario.
  5. Medición y prueba: Se probaron búsquedas y ordenamientos, observando resultados y tiempos.
  6. Documentación: Se documentó el código, se elaboró un README y se realizó este informe.
- 

#### 5. Resultados Obtenidos

- Búsqueda: La búsqueda binaria resultó mucho más rápida que la lineal, aunque requiere la lista ordenada previamente.
- Ordenamiento: Quick Sort fue claramente más rápido y eficiente que Bubble Sort, especialmente en listas de más de 100 elementos.
- El sistema simulado permitió ver la utilidad práctica de estos algoritmos en un entorno realista de gestión de productos.

---

## 6. Conclusiones

Este trabajo permitió comprender la importancia de aplicar algoritmos eficientes en la gestión de datos. En situaciones como inventarios, catálogos o sistemas de ventas, una búsqueda rápida y un orden adecuado son fundamentales para optimizar tiempos de respuesta y mejorar la experiencia del usuario. Además, se reforzaron habilidades de programación y análisis algorítmico, útiles para futuros proyectos más complejos.

---

## 7. Bibliografía

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
  - Python Software Foundation. (2024). Official Python documentation.  
<https://docs.python.org/3/>
  - GeeksforGeeks. (2024). Searching and Sorting Algorithms
  - Universidad Tecnológica Nacional (2025). Material de búsqueda y Ordenamiento en Programación
  - Universidad Tecnológica Nacional (2025). Material de análisis de Algoritmos: Fundamentos y Técnicas de Optimización
- 

## 8. Anexos

### A. Captura de ejecución del código

Busqueda lineal en lista de 50 productos desordenados

```
=====
===== INVENTARIO DE PRODUCTOS =====
=====
Seleccione una opción:
0. Ver el inventario de productos
1. Búsqueda Lineal por nombre
2. Búsqueda Binaria por nombre
3. Ordenamiento por precio (Bubble Sort)
4. Ordenamiento por stock (Quick Sort)
5. Salir
-----
Selecciona una opción: 1

=====
Nombre del producto a buscar: zapatos
Resultado: 1002 - Zapatos | Stock: 7 | $6412
Tiempo de búsqueda: 0.000013 segundos
=====

Presione Enter para continuar...|
```

Busqueda binaria en lista de 50 productos ordenados

```
=====
===== INVENTARIO DE PRODUCTOS =====
=====
Seleccione una opción:
0. Ver el inventario de productos
1. Búsqueda Lineal por nombre
2. Búsqueda Binaria por nombre
3. Ordenamiento por precio (Bubble Sort)
4. Ordenamiento por stock (Quick Sort)
5. Salir
-----
Selecciona una opción: 2

=====
Nombre del producto a buscar: zapatos
Resultado: 1002 - Zapatos | Stock: 7 | $6412
Tiempo de búsqueda: 0.000014 segundos

Presione Enter para continuar...|
```

Ordenamiento por Bubble sort(precio) en lista de 50 productos



```
=====
===== INVENTARIO DE PRODUCTOS =====
=====
Seleccione una opción:
0. Ver el inventario de productos
1. Búsqueda Lineal por nombre
2. Búsqueda Binaria por nombre
3. Ordenamiento por precio (Bubble Sort)
4. Ordenamiento por stock (Quick Sort)
5. Salir
-----
Selecciona una opción: 3

=====
Ordenando productos por precio...
Productos ordenados. Tiempo: 0.000078 segundos
Archivo generado: ordenado.csv
=====

Presione Enter para continuar...|
```

Ordenamiento por Quick Sort(stock) en lista de 50 productos

```
=====
===== INVENTARIO DE PRODUCTOS =====
=====
Seleccione una opción:
0. Ver el inventario de productos
1. Búsqueda Lineal por nombre
2. Búsqueda Binaria por nombre
3. Ordenamiento por precio (Bubble Sort)
4. Ordenamiento por stock (Quick Sort)
5. Salir
-----
Selecciona una opción: 4

=====
Ordenando productos por stock...
Productos ordenados. Tiempo: 0.000052 segundos
Archivo generado: ordenado.csv
=====

Presione Enter para continuar...|
```

B. Captura de los archivos con funciones ([Ordenamiento.py](#)) - ([busqueda.py](#))

[Ordenamiento.Py](#)

```
import csv

# Clase Producto nos permite crear objetos de tipo Producto con atributos id, nombre, stock y precio.
#facilitando la manipulación de los datos de los productos.
Windsurf: Refactor | Explain | Qodo Gen: Options | Test this class
class Producto:
    Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this method
    def __init__(self, id, nombre, stock, precio):
        self.id = id
        self.nombre = nombre
        self.stock = stock
        self.precio = precio

    # __repr__ es un método especial que devuelve una cadena con el id, nombre, stock y precio del producto.
    Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this method
    def __repr__(self):
        return f"{self.id} - {self.nombre} | Stock: {self.stock} | ${self.precio}"
Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
def cargar_productos_csv(ruta):
    productos = []
    #with open abre el archivo CSV en modo lectura y newline='' asegura que no haya problemas con los saltos de línea.
    #csv.DictReader lee el archivo CSV y convierte cada fila en un diccionario.
    with open(ruta, newline='') as file:
        #csv.DictReader lee el archivo CSV y convierte cada fila en un diccionario.
        reader = csv.DictReader(file)
        for row in reader:
            producto = Producto(
                id=int(row['ID']),
                nombre=row['Nombre'],
                stock=int(row['Stock']),
                precio=int(row['Precio'])
            )
            productos.append(producto)
    return productos


Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
def guardar_productos_csv(productos, ruta):
    #with open abre el archivo CSV en modo escritura y newline='' asegura que no haya problemas con los saltos de línea.
    with open(ruta, mode='w', newline='') as file:
        #csv.writer crea un objeto escritor que permite escribir en el archivo CSV.
        writer = csv.writer(file)
        writer.writerow(['ID', 'Nombre', 'Stock', 'Precio'])
        for p in productos:
            writer.writerow([p.id, p.nombre, p.stock, p.precio])
```

[Busqueda.py](#)

```
1 # Búsqueda lineal por nombre
Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
2 def busqueda_lineal(productos, nombre_buscado):
3     nombre_buscado = nombre_buscado.lower()
4     for producto in productos:
5         if producto.nombre.lower() == nombre_buscado:
6             return producto
7     return None
8
9 # Búsqueda binaria (requiere lista ordenada por nombre)
Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
10 def busqueda_binaria(productos, nombre_buscado):
11     nombre_buscado = nombre_buscado.lower()
12     #esta función asume que la lista de productos ya está ordenada por nombre
13     productos = sorted(productos, key=lambda x: x.nombre.lower()) # Ordenar ignorando mayúsculas/minúsculas
14     izq = 0
15     der = len(productos) - 1
16     while izq <= der:
17         medio = (izq + der) // 2
18         nombre_medio = productos[medio].nombre.lower() # Normalizo a minúsculas
19         if nombre_medio == nombre_buscado:
20             return productos[medio]
21         elif nombre_medio < nombre_buscado:
22             izq = medio + 1
23         else:
24             der = medio - 1
25     return None
26
27 # Bubble Sort por precio
Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
28 def bubble_sort(productos):
29     n = len(productos)
30     for i in range(n):
31         for j in range(0, n-i-1):
32             if productos[j].precio > productos[j+1].precio:
33                 productos[j], productos[j+1] = productos[j+1], productos[j]
34
35 # Quick Sort por stock
Windsurf: Refactor | Explain | Generate Docstring | X | Qodo Gen: Options | Test this function
36 def quick_sort(productos):
37     if len(productos) <= 1:
38         return productos
39     pivote = productos[0]
40     menores = [p for p in productos[1:] if p.stock <= pivote.stock]
41     mayores = [p for p in productos[1:] if p.stock > pivote.stock]
42     return quick_sort(menores) + [pivote] + quick_sort(mayores)
43
```

## C. Código fuente

Ver en el repositorio público de GitHub:

 <https://github.com/Gastuzar/algoritmos-inventario.git>

## D. Video Explicativo

Ver en youtube:

 <https://www.youtube.com/watch?v=BfF1tXyxVvAw>