Trabajo integrador:

ALGORITMOS DE BÚSQUEDA Y ORDENAMIENTO

Alumnos

Veronica Guirin

Rosario Iturralde

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación 1

Docente Titular

Ariel Enferrel

Docente Tutor

Maximiliano Sar Fernández

9 de junio de 2025

Alumnos:

Rosario Iturralde – rosario_iturralde@hotmail.com

Guirin Veronica – guirinveronica@outlook.es

Índice

1. Introducción

Los algoritmos de **búsqueda y ordenamiento** son fundamentales en la programación y la ciencia de la computación, ya que permiten organizar y recuperar información de manera eficiente. Su estudio es esencial para optimizar el rendimiento de aplicaciones que manejan grandes volúmenes de datos, como motores de búsqueda, bases de datos y sistemas de recomendación. Con el crecimiento exponencial de datos, la selección de algoritmos adecuados puede marcar la diferencia entre un sistema rápido y uno ineficiente. Mientras que algoritmos como **QuickSort** y **Binary Search** ofrecen un rendimiento óptimo en la mayoría de los casos, otros como **Bubble Sort** o **Linear Search** son menos eficientes, pero más simples de implementar. Este trabajo explora algunos de los principales algoritmos de búsqueda y ordenamiento en un caso práctico que utiliza una base de datos de precipitaciones.

Objetivo del Trabajo

El objetivo principal de este trabajo fue comprender e implementar los algoritmos de búsqueda y ordenamiento más utilizados en el lenguaje de programación (Python) para evaluar su rendimiento e identificar ventajas y desventajas al usar cada algoritmo.

2. Marco Teórico

Algoritmos de Búsqueda

Los algoritmos de búsqueda permiten encontrar un elemento dentro de una estructura de datos. Entre los más relevantes se encuentran:

Búsqueda Lineal (Linear Search)

- ✓ **Complejidad:** *O(n)*, ineficiente para grandes conjuntos de datos.
- ✓ **Funcionamiento:** Recorre secuencialmente una lista hasta encontrar el elemento.

Búsqueda Binaria (Binary Search)

- ✓ **Complejidad:** *O(log n)*, requiere datos ordenados.
- ✓ **Funcionamiento:** Divide repetidamente el conjunto de datos a la mitad para localizar el valor.

Algoritmos de Ordenamiento

Los algoritmos de ordenamiento organizan datos en un orden específico (ascendente/descendente). Algunos de los más estudiados son:

Bubble Sort

✓ **Complejidad:** *O(n²)*, poco eficiente, pero fácil de entender.

✓ **Funcionamiento:** Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto.

QuickSort

✓ **Complejidad:** *O(n log n)* en promedio, muy eficiente.

✓ **Funcionamiento:** Utiliza un "pivote" para dividir el arreglo en subconjuntos y ordenarlos recursivamente.

Selection Sort

✓ **Complejidad:** *O(n²) (simple pero ineficiente para grandes volúmenes).*

✓ **Funcionamiento:** Recorre el arreglo, selecciona el mínimo y lo intercambia con el primer elemento no ordenado. Repite el proceso hasta ordenar todo el arreglo.

Aplicaciones en la Vida Real

Los algoritmos de búsqueda y ordenamiento tienen múltiples aplicaciones prácticas en el desarrollo de software. La búsqueda binaria, por ejemplo, se utiliza ampliamente en sistemas de bases de datos indexadas, ya que permite encontrar registros de forma muy eficiente cuando los datos están ordenados. Por su parte, QuickSort es uno de los algoritmos más veloces en la práctica y se encuentra implementado en muchas librerías estándar, como el método list.sort() de Python. Finalmente, la búsqueda lineal, aunque menos eficiente, sigue siendo útil en listas pequeñas o cuando los datos no están ordenados, ya que no requiere preparación previa ni estructuras adicionales.

3. Caso Práctico

El programa desarrollado permite evaluar el comportamiento de algoritmos de búsqueda y ordenamiento aplicados a bases de datos meteorológicas. En este caso, se trabaja con dos conjuntos de datos reales: una base con registros de precipitaciones correspondientes a 10 partidos de la Provincia de Buenos Aires durante los últimos 5 años, y otra más reducida con datos de 5 partidos. Estas bases se utilizan para probar el rendimiento del sistema en distintos volúmenes de información, permitiendo realizar búsquedas por localidad o año, y ordenar los registros por cantidad de lluvia, año o nombre de localidad. Esta práctica no solo permite comparar la eficiencia de los algoritmos implementados, sino también visualizar cómo responden ante diferentes cantidades de datos.

Código del programa:

```
#Algoritmo de busqueda y ordenamiento elaborado pata tp programación 1. Guirin
- Iturralde.
import csv #Importa el módulo estándar csv que permite leer y escribir archivos
en formato csv.
import os # Se importa para verificar que los archivos .csv estén realmente en
la carpeta.
import time # Importa el módulo time, que permite trabajar con el tiempo del
sistema.


# Función para leer el archivo CSV
def cargar_datos(nombre_archivo):
    if not os.path.exists(nombre_archivo):
```

```
        print(f"Error: El archivo {nombre_archivo} no existe.")
        return []
    with open(nombre_archivo, encoding="latin1") as f:
        lector = csv.DictReader(f, delimiter=';')
        datos = list(lector)
        for fila in datos:
            fila["Año"] = int(fila["Año"])
            fila["Lluvia (mm)"] = float(fila["Lluvia (mm)"].replace(",", "."))
        return datos

# Función para imprimir datos en formato legible
def imprimir_fila(fila):
    print(f"Año: {fila['Año']}, Localidad: {fila['Localidad']}, Lluvia (mm):
{fila['Lluvia (mm)']:.2f}")

# Función para guardar resultados en CSV
def guardar_resultados(data, nombre_archivo):
    with open(nombre_archivo, "w", newline='', encoding="latin1") as f:
        campos = ["Año", "Localidad", "Lluvia (mm)"]
        escritor = csv.DictWriter(f, fieldnames=campos, delimiter=';')
        escritor.writeheader()
        for fila in data:
            escritor.writerow(fila)
    print(f"Resultados guardados en {nombre_archivo}.")
```

```python
# Funciones de ordenamiento
def ordenar_por_lluvia(data):
    data = data.copy()
    for i in range(len(data)):
        max_idx = i
        for j in range(i+1, len(data)):
            if data[j]["Lluvia (mm)"] > data[max_idx]["Lluvia (mm)"]:
                max_idx = j
        data[i], data[max_idx] = data[max_idx], data[i]
    return data

def ordenar_por_año(data):
    data = data.copy()
    n = len(data)
    for i in range(n):
        for j in range(0, n-i-1):
            if data[j]["Año"] > data[j+1]["Año"]:
                data[j], data[j+1] = data[j+1], data[j]
    return data

def ordenar_por_localidad(data):
    if len(data) <= 1:
        return data
    pivote = data[len(data)//2]["Localidad"]
    menores = [x for x in data if x["Localidad"] < pivote]
```

```
    iguales = [x for x in data if x["Localidad"] == pivote]
    mayores = [x for x in data if x["Localidad"] > pivote]
    return ordenar_por_localidad(menores) + iguales +
ordenar_por_localidad(mayores)

# Funciones de búsquedas
def buscar_por_localidad(data, localidad):
    return [fila for fila in data if fila["Localidad"].lower() ==
localidad.lower()]

def buscar_por_año(data, año):
    return [fila for fila in data if fila["Año"] == año]

# Menú interactivo
def menu():
    print("\nSistema de Precipitaciones")
    print("1. Ordenar por lluvia (mm)")
    print("2. Ordenar por año")
    print("3. Ordenar por localidad")
    print("4. Buscar por localidad")
    print("5. Buscar por año")
    print("6. Cambiar archivo CSV")
    print("7. Salir")

#Programa principal
```

```
# Cargar dos archivos al inicio
print("Carga de datos:")
archivo1 = input("Ingrese el nombre del primer archivo CSV: ")
archivo2 = input("Ingrese el nombre del segundo archivo CSV: ")

datos1 = cargar_datos(archivo1)
datos2 = cargar_datos(archivo2)

datos_actuales = datos1 if datos1 else datos2

while True:
    print(f"\nActualmente trabajando con: {archivo1 if datos_actuales == datos1
else archivo2}")
    menu()
    opcion = input("Seleccione una opción: ")

    if opcion == "1":
        inicio = time.time()
        ordenado = ordenar_por_lluvia(datos_actuales)
        fin = time.time()
        print(f"\nDatos ordenados por lluvia (mayor a menor) en {fin -
inicio:.4f} segundos:")
        for fila in ordenado[:10]:
            imprimir_fila(fila)
        print("... Mostrando primeros 10 resultados.")
```

```
    elif opcion == "2":
        inicio = time.time()
        ordenado = ordenar_por_año(datos_actuales)
        fin = time.time()
        print(f"\nDatos ordenados por año en {fin - inicio:.4f} segundos:")
        for fila in ordenado[:10]:
            imprimir_fila(fila)

    elif opcion == "3":
        inicio = time.time()
        ordenado = ordenar_por_localidad(datos_actuales)
        fin = time.time()
        print(f"\nDatos ordenados por localidad en {fin - inicio:.4f}
segundos:")
        for fila in ordenado[:10]:
            imprimir_fila(fila)

    elif opcion == "4":
        loc = input("Ingrese el nombre de la localidad: ")
        inicio = time.time()
        resultados = buscar_por_localidad(datos_actuales, loc)
        fin = time.time()
        print(f"\n{len(resultados)} resultados encontrados en {fin -
inicio:.4f} segundos.")
```

```
        if resultados:
            for r in resultados[:10]:
                imprimir_fila(r)
            print("... Mostrando primeros 10 resultados.")
            if input("¿Desea guardar los resultados? (s/n): ").lower() == "s":
                guardar_resultados(resultados, "resultados_localidad.csv")
        else:
            print("No se encontraron registros para esa localidad.")


    elif opcion == "5":
        try:
            año = int(input("Ingrese el año: "))
            inicio = time.time()
            resultados = buscar_por_año(datos_actuales, año)
            fin = time.time()
            print(f"\n{len(resultados)} resultados encontrados en {fin -
inicio:.4f} segundos.")
            if resultados:
                for r in resultados[:10]:
                    imprimir_fila(r)
                print("... Mostrando primeros 10 resultados.")
                if input("¿Desea guardar los resultados? (s/n): ").lower() ==
"s":
                    guardar_resultados(resultados, "resultados_año.csv")
            else:
```

```
                print("No hay datos para ese año.")
        except ValueError:
            print("Ingrese un año válido.")

    elif opcion == "6":
        print("1. Cambiar a primer archivo")
        print("2. Cambiar a segundo archivo")
        cambio = input("Seleccione una opción: ")
        if cambio == "1" and datos1:
            datos_actuales = datos1
        elif cambio == "2" and datos2:
            datos_actuales = datos2
        else:
            print("Archivo no disponible.")

    elif opcion == "7":
        print("¡Hasta luego!")
        break

    else:
        print("Opción inválida.")
```

4. Metodología Utilizada

4.1. Importación de módulos

Se importan las bibliotecas csv, os y time para trabajar con archivos, verificar su existencia y medir tiempos de ejecución.

4.2 Carga de archivos CSV

El usuario elige entre dos archivos .csv con datos de precipitaciones de partidos de la Provincia de Buenos Aires. Se leen usando csv.DictReader.

4.3. Conversión de datos

Los valores leídos (como "Año" y "Lluvia (mm)") se convierten a los tipos adecuados (int y float) para poder trabajar con ellos correctamente.

4.4. Implementación de algoritmos de ordenamiento

Se incluyen tres métodos:

Selection Sort para ordenar por cantidad de lluvia (de mayor a menor).

Bubble Sort para ordenar por año (de menor a mayor).

QuickSort para ordenar por nombre de localidad (alfabéticamente).

4.5. Implementación de búsquedas

Se puede buscar por:

Localidad (comparación directa de cadenas).

Año (comparación numérica).

4.6. Medición de tiempos de ejecución

Se mide cuánto tarda cada búsqueda u ordenamiento utilizando time.time() antes y después de la operación.

4.7. Menú interactivo

El usuario accede a un menú en consola que permite:

- ✓ Elegir el tipo de ordenamiento o búsqueda.
- ✓ Ingresar valores (como año o localidad) para filtrar datos.
- ✓ Salir del programa.
- ✓ Impresión de resultados
- ✓ Se muestran en pantalla los registros procesados, ordenados o filtrados según lo solicitado.

5. **Resultados**

Este programa fue puesto a prueba con dos bases de datos de precipitaciones del sudeste bonaerense. Cada archivo .csv tiene como columnas Fecha, Mes, Año, Localidad y Lluvia (mm). Ambas bases de datos tienen como objetivo almacenar las precipitaciones, es decir las lluvias, desde junio de 2020 y actualmente posee datos hasta mayo de 2025. El archivo lluvias1.csv posee datos de 10 partidos: Balcarce, Benito Juárez, Gral. Alvarado, Gral. La Madrid, Gral. Pueyrredón, Laprida, Lobería, Necochea, Olavarría y Tandil, y tiene 18225 filas. El archivo lluvias2.csv posee datos de 10 partidos: Balcarce, Benito Juárez, Gral. Alvarado, Gral. La Madrid, Gral. Pueyrredón, y tiene 9107 filas. Una vez desarrollado el programa según los pasos descritos en la metodología utilizada, se puso a prueba y el primer inconveniente fue revisar problemas que tenía la base de datos, como comas en vez de puntos y otros caracteres especiales que generaron errores en las primeras pruebas. En ese sentido trabajar con una base de datos real que es actualizada cada 10 o 15 días por diferentes personas nos resultó muy enriquecedor, a diferencia de si hubiésemos trabajado con una base de datos simulada.

En cuanto al tiempo de ejecución pudimos comprobar como el tiempo de los procesos era considerablemente menor en la base de datos de lluvias2.cvs, que posee la mitad de los datos que la base de datos lluvias1.csv (ver Anexo).

Para finalizar queremos mencionar algunas consideraciones relevantes de cada función elaborada:

5.1. buscar_por_localidad: pertenece al tipo de **búsqueda lineal**. Para su funcionamiento recorre cada fila de la lista. Compara el valor de "Localidad" con el texto ingresado. Si encuentra coincidencia, la agrega a una nueva lista. Como ventaja es muy simple de implementar y funciona incluso si los datos no están ordenados, cuestión que resulta muy útil en esta base de datos por la manera que se va actualizando. Cómo desventaja es poco eficiente con muchos datos: complejidad O(n), recorre todos los elementos, aunque la coincidencia esté al principio.

5.2. buscar_por_año: pertenece al tipo de búsqueda lineal (secuencial). Es exactamente igual que la anterior, pero compara por "Año". Revisa todos los elementos uno por uno. Una cuestión para destacar es que, si los datos estuvieran ordenados por localidad o por año, podrías usar **búsqueda binaria**, que es mucho más eficiente.

5.3. ordenar_por_lluvia pertenece al tipo de búsqueda **Selection Sort**. Funciona recorriendo la lista buscando el máximo valor restante. Luego, lo coloca en su posición correspondiente (en este caso, al principio, para orden descendente). Repite este proceso para cada posición de la lista. Es un tipo de algoritmo de ordenamiento por selección. Como desventaja se puede mencionar que puede cambiar el orden de elementos iguales, pero es fácil de entender e implementar.

5.4. ordenar_por_año pertenece al tipo de búsqueda **Bubble Sort**. Para funcionar compara elementos adyacentes y los intercambia si están en el orden incorrecto. En cada pasada, el elemento más grande "burbujea" hasta el final. Se considera un algoritmo de ordenamiento por intercambio. Una de sus principales ventajas es que es muy simple y fácil de optimizar con banderas.

5.5. ordenar_por_localidad: pertenece al tipo de búsqueda **QuickSort**. Para funcionar elige un pivote (en este caso, el del medio). Divide la lista en tres: menores al pivote, iguales al pivote, mayores al pivote. Aplica el mismo proceso recursivamente a los menores y mayores. Es un algoritmo que aplica ordenamiento por división y conquista. El peor caso sería la lista ya ordenada y mal pivote. Como ventaja

es muy rápido en la práctica, pero al usa recursión lo que podría causar problemas con listas muy grandes.

6. Conclusiones

El desarrollo de este trabajo nos permitió comprender de manera práctica cómo funcionan distintos algoritmos de búsqueda y ordenamiento, evaluando sus rendimientos en el manejo de bases de datos reales con información meteorológica. A través de la implementación en Python, se pudo evidenciar que la elección del algoritmo tiene un impacto directo en la eficiencia del procesamiento, especialmente cuando se trabaja con grandes volúmenes de datos.

Se comprobó que los algoritmos más simples, como la búsqueda lineal o el ordenamiento por burbujeo, son útiles en contextos reducidos o cuando se requiere facilidad de implementación, pero presentan limitaciones de rendimiento en bases más extensas. Por el contrario, algoritmos más sofisticados como QuickSort o la búsqueda binaria (potencialmente aplicable si los datos estuvieran ordenados) ofrecen mejores tiempos de ejecución, aunque requieren mayor preparación o condiciones específicas.

Además, el uso de datos reales introdujo desafíos adicionales, por lo que sería una mejora agregar al programa el tratamiento de formatos inconsistentes y caracteres especiales. Esto enriqueció el proceso de desarrollo y nos puso a pensar en futuras soluciones más robustas y adaptadas a entornos dinámicos.

Esta experiencia práctica no solo facilitó el entendimiento de los algoritmos, sino que también brindó herramientas valiosas para abordar problemas reales en programación y análisis de datos.

7. Bibliografía

https://www.fing.edu.uy/tecnoinf/mvd/cursos/prinprog/material/teo/prinprog-teorico11.pdf .
Consulta 1 de junio de 2025.

https://gc.scalahed.com/recursos/files/r161r/w24810w/Algoritmos_de_busqueda_y_ordenamiento.pdf . Consulta 1 de junio de 2025.

Material Unidad búsqueda y ordenamiento de la Tecnicatura Universitaria en Programación. Catedra Programación 1. Universidad Tecnológica Nacional.

8. Anexos

Datos ordenados por lluvia archivo lluvias1.cvs

```
Datos ordenados por lluvia (mayor a menor) en 60.3566 segundos:
Año: 2021, Localidad: Laprida, Lluvia (mm): 230.00
Año: 2023, Localidad: Laprida, Lluvia (mm): 126.00
Año: 2025, Localidad: Gral Lamadrid, Lluvia (mm): 118.70
Año: 2021, Localidad: Gral Lamadrid, Lluvia (mm): 112.50
Año: 2025, Localidad: Laprida, Lluvia (mm): 105.70
Año: 2021, Localidad: Gral Lamadrid, Lluvia (mm): 99.50
```

Datos ordenados por año archivo lluvias1.cvs

```
Datos ordenados por año en 64.9993 segundos:
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
```

Datos ordenados por localidad archivo lluvias1.cvs

```
Datos ordenados por localidad en 0.1182 segundos:
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
```

Búsqueda por localidad archivo lluvias1.cvs

```
Ingrese el nombre de la localidad: Balcarce

1821 resultados encontrados en 0.0160 segundos.
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
```

Búsqueda por año archivo lluvias1.cvs

```
Ingrese el año: 2023

3650 resultados encontrados en 0.0000 segundos.
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 8.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
```

Datos ordenados por lluvia archivo lluvias2.cvs

```
Datos ordenados por lluvia (mayor a menor) en 13.5887 segundos:
Año: 2025, Localidad: Gral Lamadrid, Lluvia (mm): 118.70
Año: 2021, Localidad: Gral Lamadrid, Lluvia (mm): 112.50
Año: 2021, Localidad: Gral Lamadrid, Lluvia (mm): 99.50
Año: 2023, Localidad: Balcarce, Lluvia (mm): 98.00
Año: 2025, Localidad: Benito Juarez, Lluvia (mm): 92.00
Año: 2023, Localidad: Balcarce, Lluvia (mm): 89.00
```

Datos ordenados por año archivo lluvias2.cvs

```
Datos ordenados por año en 19.8007 segundos:
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2020, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
```

Datos ordenados por localidad archivo lluvias2.cvs

```
Datos ordenados por localidad en 0.0156 segundos:
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
```

Búsqueda por localidad archivo lluvias2.cvs

```
Ingrese el nombre de la localidad: Balcarce

1821 resultados encontrados en 0.0053 segundos.
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
Año: 2024, Localidad: Balcarce, Lluvia (mm): 0.00
```

Búsqueda por año archivo lluvias2.cvs

```
Ingrese el año: 2023

1824 resultados encontrados en 0.0000 segundos.
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 8.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
Año: 2023, Localidad: Gral Pueyrredon, Lluvia (mm): 0.00
```