UNIVERSIDAD TECNOLÓGICA NACIONAL

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN A DISTANCIA

PROGRAMACIÓN I

Algoritmos de Búsqueda y Ordenamiento

Alumnos:
ANGELELLI, RODRIGO - rodrigoangelelli3@gmail.com
SCHNEIDER, ASTRID - astridelizabethschneider@gmail.com

Comisión N°24

Profesor: Bruselario, Sebastián
Tutor: Gubiotti, Flor

Fecha de Entrega: 09/06/2025

Índice

INTRODUCCIÓN

"Lo que no se puede medir, no se puede mejorar."

— Lord Kelvin

El presente trabajo aborda el estudio de los algoritmos de búsqueda y ordenamiento, dos herramientas fundamentales en el área de la programación. La elección de este tema surge del interés por comprender cómo se procesan y organizan los datos dentro de un programa, ya que estas operaciones son esenciales en el desarrollo de software eficiente. Buscar elementos dentro de una estructura y ordenarlos según determinados criterios son tareas frecuentes que impactan directamente en el rendimiento y la escalabilidad de las aplicaciones.

En programación, la forma en que se manipulan los datos tiene consecuencias significativas sobre la **velocidad de ejecución**, el **uso de memoria** y la **capacidad de adaptación a grandes volúmenes de información**. Elegir un algoritmo adecuado para buscar u ordenar no solo mejora el tiempo de respuesta de un programa, sino que también reduce la carga computacional, optimiza recursos y permite que el sistema funcione correctamente incluso cuando crece la cantidad de datos.

El objetivo de este trabajo es analizar los principales algoritmos de búsqueda y ordenamiento utilizados en programación, describir su funcionamiento, evaluar su eficiencia mediante la notación Big O y comprender en qué contextos es conveniente aplicar cada uno. A través de este análisis, se espera obtener una visión clara sobre cómo estas herramientas contribuyen a resolver problemas concretos en el desarrollo de software, favoreciendo la escritura de programas más rápidos, ordenados y escalables.

MARCO TEÓRICO

1 - BÚSQUEDA Y ORDENAMIENTO: FUNCIONES DIFERENTES, OBJETIVOS COMUNES

En programación, los algoritmos de **búsqueda** y **ordenamiento** cumplen funciones diferentes pero complementarias. La **búsqueda** consiste en localizar un elemento específico dentro de una estructura de datos, como por ejemplo una lista o un arreglo. El objetivo es encontrar, si existe, la posición o el valor deseado, utilizando un método que puede ir desde una simple exploración secuencial hasta técnicas más sofisticadas como la búsqueda binaria.

Por otro lado, el **ordenamiento** se refiere a reorganizar un conjunto de elementos según un criterio definido, como el orden numérico o alfabético. Un conjunto de datos ordenado no solo facilita la comprensión visual, sino que también permite aplicar algoritmos de búsqueda más eficientes, como es el caso de la búsqueda binaria, que requiere que los datos estén previamente ordenados.

En resumen, mientras que la **búsqueda** apunta a encontrar un valor dentro de un conjunto, el **ordenamiento** tiene como objetivo reorganizar ese conjunto para que cumpla una determinada condición de orden. Ambas operaciones son fundamentales y, en muchos casos, se complementan para mejorar el rendimiento y la funcionalidad de los programas.

2 - FORMAS DE ENCONTRAR LO QUE BUSCAMOS

En programación, los algoritmos de búsqueda permiten **localizar un valor dentro de una estructura de datos**, como listas, tuplas y diccionarios. Elegir el algoritmo adecuado depende del tamaño del conjunto de datos, de si está ordenado o no, y del costo en tiempo y memoria que se desea asumir.

Búsqueda lineal (o secuencial):

La **búsqueda lineal** es el algoritmo más básico para encontrar un valor dentro de una estructura de datos, como una lista o un arreglo. Se llama “lineal” porque recorre los elementos uno por uno, de forma secuencial, desde el principio hasta el final, comparando cada elemento con el valor que se quiere encontrar.

Este método **no requiere que los datos estén ordenados**, lo cual lo hace útil en muchos casos prácticos. Sin embargo, es **poco eficiente en listas grandes**, ya que en el peor de los casos tiene que recorrer todos los elementos.

¿Cómo funciona?

1. Se empieza desde el primer elemento de la lista.
2. Se compara ese elemento con el valor buscado.
3. Si coinciden, se devuelve la posición (o el valor).
4. Si no coinciden, se pasa al siguiente elemento.
5. Si se llega al final sin encontrar el valor, se concluye que no está en la lista.

Complejidad

- **Mejor caso:** O(1) (si el elemento está en la primera posición)
- **Peor caso:** O(n) (si está al final o no está)

Ventajas

- Sencilla de implementar.
- Funciona con cualquier tipo de lista (ordenada o no).

Desventajas

- Lenta para listas grandes.
- No aprovecha ninguna propiedad de los datos.

Búsqueda Binaria:

La búsqueda binaria es un algoritmo **eficiente** para encontrar un elemento en una **lista ordenada**. Funciona dividiendo repetidamente el espacio de búsqueda a la mitad, descartando la mitad en la que el elemento buscado no puede estar.

Requisitos:

- La lista debe estar ordenada (de menor a mayor).
- Si la lista no está ordenada, primero hay que ordenarla (por ejemplo, usando métodos como .sort() en Python).

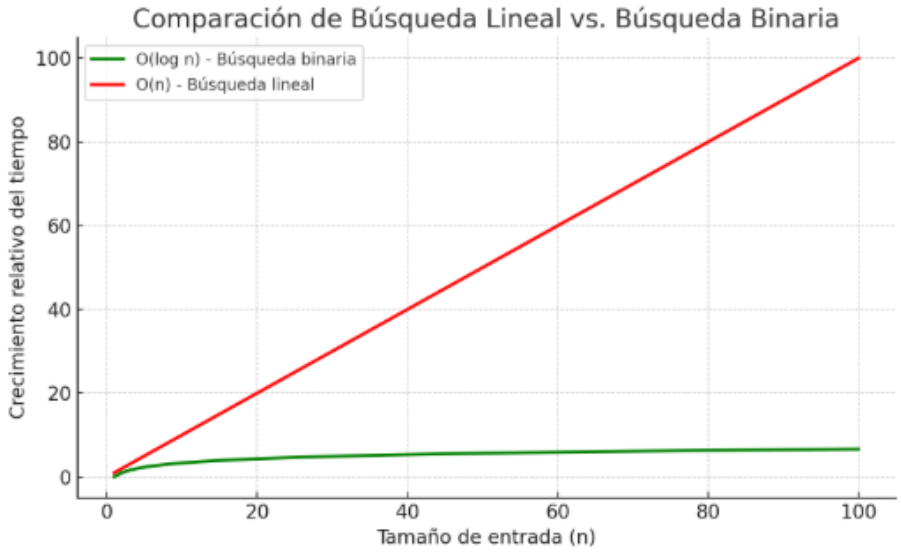
Cómo funciona:

Supongamos que queremos buscar el número 15 en la lista ordenada: [2, 5, 8, 12, 15, 19, 23, 28, 31]

1. Miramos el elemento del medio (índice 4, valor 15).
2. Si es igual al objetivo, ¡listo, lo encontramos!
3. Si el objetivo es menor que el elemento del medio, descartamos la mitad derecha.
4. Si es mayor, descartamos la mitad izquierda.
5. Repetimos el proceso en la mitad seleccionada hasta encontrar el elemento o que no queden elementos.

Comparación entre búsqueda lineal y búsqueda binaria:

Característica	Búsqueda Lineal	Búsqueda Binaria
Requisito	Lista puede estar desordenada	Lista debe estar ordenada
Método	Recorre elemento por elemento	Divide repetidamente la lista a la mitad
Complejidad temporal - notación Big O	O(n) (lineal)	O(log n) (logarítmica)
Eficiencia	Menos eficiente para listas grandes	Muy eficiente para listas grandes
Uso de memoria	No requiere memoria extra	No requiere memoria extra
Facilidad de implementación	Muy simple	Un poco más compleja que la lineal
Aplicaciones comunes	Listas pequeñas o no ordenadas	Listas ordenadas, búsqueda rápida



En el siguiente enlace se puede visualizar una animación comparativa de ambos métodos :

- **https://miro.medium.com/v2/resize:fit:640/format:webp/0*wKLd_dHgFcPPJMCL.gif**

3 - EXPLORANDO ALGORITMOS DE ORDENAMIENTO

El ordenamiento de datos es una operación fundamental en programación y estructuras de datos. Organizar correctamente una colección permite realizar tareas más eficientemente, como búsquedas, comparaciones, análisis o visualizaciones. Además, muchos algoritmos dependen de que los datos estén ordenados previamente para funcionar de forma óptima (como la búsqueda binaria).

Existen múltiples algoritmos de ordenamiento, cada uno con características particulares en cuanto a complejidad temporal, uso de memoria y facilidad de implementación. Algunos están pensados para listas pequeñas y simples (como **Bubble Sort**), mientras que otros son más eficientes para grandes volúmenes de datos (como **Merge Sort** o **Quick Sort**).

Bubble Sort (Ordenamiento Burbuja):

Este método de ordenamiento se basa en recorrer el vector de datos comparando los elemento adyacentes entre sí, intercambiandolos si el elemento de la derecha es mayor o menor según el criterio que se desee emplear. Esto debe repetirse una y otra vez sobre el vector hasta dejarlo ordenado.

Se puede visualizar una animación en el siguiente enlace:

- https://tute-avalos.com/images/bubblesort.gif

Bubble Sort es un algoritmo muy fácil de entender e implementar, pero **no es eficiente para listas grandes** porque su **tiempo de ejecución crece rápido a medida que aumenta el tamaño de la lista** (es decir, su complejidad en el peor caso es $O(n^2)$).

Esto significa que si la lista tiene muchos elementos, Bubble Sort puede tardar mucho en ordenarlos, ya que tiene que hacer muchas comparaciones y cambios.

Sin embargo, para listas pequeñas o casi ordenadas, Bubble Sort puede ser razonablemente rápido, porque en el mejor caso termina en tiempo lineal O(n).

Merge Sort (Ordenamiento por mezcla):

Merge Sort es un algoritmo de ordenamiento que utiliza la estrategia **divide y vencerás**.

Esto significa que:

1. Divide la lista original en partes más pequeñas hasta que cada parte tenga un solo elemento (que está, por definición, ordenado).
2. Luego mezcla (merge) esas partes pequeñas de forma ordenada para formar listas más grandes ordenadas.
3. Finalmente, se obtendrá la lista ordenada completa.

Ejemplo visual:

Lista completa:
[38, 27, 43, 3, 9, 82, 10]
Divide:
[38, 27, 43] [3, 9, 82, 10]
Divide más:
[38] [27] [43] [3] [9] [82] [10]
Mezcla pares:
[27, 38] [43] [3, 9] [10, 82]
Mezcla más:
[27, 38, 43] [3, 9, 10, 82]
Mezcla final:
[3, 9, 10, 27, 38, 43, 82]

Se puede visualizar una animación en el siguiente enlace:

- https://miro.medium.com/v2/resize:fit:600/format:webp/0*5XcI7fZtyxSIL7TQ.gif

QuickSort (Ordenamiento rápido): Quicksort es un algoritmo de ordenamiento que también usa la estrategia divide y vencerás, pero con otra forma de dividir:

1. Elige un pivote (un elemento de la lista).
2. Reorganiza la lista para que todos los elementos menores al pivote queden a la izquierda, y todos los mayores a la derecha.
3. Luego ordena recursivamente esas dos partes.
4. Finalmente, la lista queda ordenada.

Ventajas: Quick Sort es un algoritmo rápido para ordenar listas porque, en promedio, tarda un tiempo proporcional a **n log n**.

¿Qué significa eso?

- La **n** representa que el tiempo crece según la cantidad de elementos que tengas. El **log n** (logaritmo de n) significa que a medida que duplicó la cantidad de elementos, el número de "pasos extras" solo aumenta un poco, no mucho.

Ordena en sitio (in-place): No necesita mucho espacio extra porque reordena la lista en el mismo lugar. (trabaja sobre la misma lista original, moviendo o intercambiando los elementos, en lugar de crear nuevas listas o copias grandes - no necesita usar mucha memoria extra para ordenar los datos) **Flexible:** Puede adaptarse con distintas estrategias para elegir el pivote y mejorar su rendimiento.

Desventajas:

- **Peor caso $O(n^2)$:** Cuando el pivote no divide la lista en partes iguales o casi iguales, una parte queda muy grande y la otra muy pequeña. Esto hace que el algoritmo tenga que hacer más comparaciones y recursiones, aumentando el tiempo de ejecución.
- **No es estable:** Puede cambiar el orden relativo de elementos iguales.
- **Dependencia del pivote:** Su eficiencia depende mucho de elegir un buen pivote. Si no, puede degradar el rendimiento.

Se puede visualizar una animación en el siguiente enlace:

- https://tute-avalos.com/images/quicksort.gif

Insertion Sort (Ordenamiento por inserción): Insertion Sort construye la lista ordenada de izquierda a derecha. Toma un elemento y lo inserta en el lugar correcto **dentro de los ya ordenados**.

1. Se empieza desde el segundo elemento de la lista (el primero ya se considera ordenado).
2. Se compara con los anteriores.
3. Si el elemento es menor, se desplazan los mayores una posición a la derecha.
4. Se inserta el elemento en su posición correcta.
5. Se repite hasta ordenar toda la lista.

Ventajas:

- Simple de implementar y entender.
- Eficiente para listas **muy pequeñas** o casi ordenadas.
- No requiere memoria adicional (es **in-place**).
- Estable (no cambia el orden de elementos iguales).

Desventajas

- Muy lento para listas grandes.
- Su rendimiento se degrada rápidamente cuando aumenta la cantidad de elementos.

Se puede visualizar una animación en el siguiente enlace:

- https://tute-avalos.com/images/insertionsort.gif

TimSort (ordenamiento híbrido desarrollado especialmente para Python): Timsort es un algoritmo de ordenamiento híbrido desarrollado especialmente para Python, no es un algoritmo "puro" como Bubble Sort, Insertion Sort, Merge Sort o Quicksort. En cambio, combina lo mejor de dos mundos:

- Merge Sort, que divide y conquista para lograr eficiencia en grandes volúmenes de datos.
- Insertion Sort, que es muy eficiente cuando los datos ya están parcialmente ordenados.

Se recorre la lista original y se aplica una función de clave (key) a cada elemento, los elementos se ordenan en función del valor devuelto por esa clave. El resultado es una nueva lista ordenada, sin modificar la original.

Ventajas

- Muy eficiente: complejidad O(n log n) en el peor caso.
- Estable: no altera el orden de elementos con claves iguales.
- Permite ordenar por cualquier campo.
- Fácil de usar con lambda o funciones propias.
    - lambda es una forma rápida de crear una función anónima, es decir, una función sin nombre. Se usa para funciones simples de una sola línea, cómo extraer un campo o hacer un cálculo. Se usa comúnmente en sorted() para indicar por qué campo ordenar.

Desventajas

- No permite modificar el orden "en el lugar" (in-place) como .sort().
- No permite múltiples criterios sin combinaciones adicionales.

Se puede visualizar una animación en el siguiente enlace:

- https://www.chrislaux.com/timsort

4 - ANÁLISIS DE COMPLEJIDAD ALGORÍTMICA

La notación Big O (O grande) es el estándar formal que se utiliza en ciencias de la computación para describir la complejidad algorítmica, es decir, cómo se comporta un algoritmo a medida que aumenta la cantidad de datos de entrada.

Big O no mide el tiempo exacto, sino el crecimiento de las operaciones en función del tamaño de la entrada (n), especialmente cuando n es muy grande.

¿Qué representa?

- Tiempo: ¿Cuántos pasos o comparaciones realiza el algoritmo?
- Espacio: ¿Cuánta memoria extra necesita?

Por ejemplo:

- O(n) → El número de operaciones crece proporcionalmente con la entrada.
- O(n^2) → Las operaciones crecen cuadráticamente (peor).
- O(log n) → Las operaciones crecen lentamente (muy eficiente).

¿Por qué importa esto?

- Te permite elegir el algoritmo correcto según el problema.
- Aunque dos algoritmos resuelven el mismo problema, uno puede escalar mucho mejor que otro.
- En listas chicas, todos parecen rápidos, pero en datos reales y grandes, las diferencias son enormes.

Algoritmo	Mejor Caso	Peor Caso	Promedio	Estabilidad	Uso de Memoria	Ventajas Principales	Desventajas Principales
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	Estable	In-place	Muy simple de entender e implementar.	Muy ineficiente para listas grandes.
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Estable	In-place	Es eficiente con listas pequeñas o casi ordenadas.	Ineficiente con listas desordenadas y grandes.
Merge Sort	O(n log n)	O(n log n)	O(n log n)	Estable	Usa memoria extra	Muy rápido y confiable. Ideal para grandes volúmenes.	Usa más memoria (no es in-place).
Quick Sort	O(n log n)	$O(n^2)$	O(n log n)	No siempre	In-place	Muy rápido y eficiente en la práctica.	Peor caso lento; no es estable.
Tim Sort	O(n)	O(n log n)	O(n log n)	Estable	Usa memoria extra	Extremadamente eficiente en la práctica para datos del mundo real.	usa memoria extra (no es "in-place")

Un ejemplo concreto:

Ordenar una lista de 1000 elementos:

- Merge Sort (O(n log n)) → ~10.000 operaciones
- Bubble Sort ($O(n^2)$) → ~1.000.000 operaciones

Mediante el análisis de su **complejidad algorítmica**, expresada en **notación Big O**, fue posible comparar objetivamente la eficiencia de cada método.

Se observó que algoritmos simples como **Bubble Sort** o **Insertion Sort**, aunque fáciles de entender e implementar, resultan poco eficientes para grandes volúmenes de datos debido a su comportamiento cuadrático ($O(n^2)$). En contraste, algoritmos como **Merge Sort** y **Quick Sort**, con una complejidad promedio de O(n log n), ofrecen un rendimiento notablemente superior en la práctica.

Además, se destacó que la **estabilidad**, el uso de memoria (**in-place**) y la previsibilidad del rendimiento son factores clave al momento de elegir un algoritmo según el contexto.

Comprender la **complejidad temporal** y el análisis mediante **Big O** permite no solo mejorar el rendimiento de los programas, sino también tomar decisiones fundamentadas en la optimización de recursos, lo cual es esencial en el desarrollo de software eficiente y escalable.

Caso Práctico

Un club deportivo necesita un sistema que le permita gestionar la información de sus deportistas de forma eficiente. Actualmente, los datos se manejan de forma manual, lo que dificulta tareas básicas como buscar un atleta por DNI, organizar listas por edad o generar reportes por disciplina. Para resolver esta situación, se desarrolló una simulación de sistema de gestión de deportistas, donde se aplican algoritmos de búsqueda (lineal y binaria) y ordenamiento (Timsort, Bubble Sort, entre otros) para organizar y consultar la información almacenada.

Código en Python:

```python
#Importamos las librerías a utilizar.
import random
import time

#Datos base ficticios (evitar crear una base de datos manualmente)
nombres = ["Lucía", "Tomás", "María", "Juan", "Sofía", "Mateo", "Camila",
"Agustín","Valentina","Benjamín","Pedro","Jaime","Lucas","Sandra","Esteban","Ez
equiel","Milagros"]
apellidos = ["González", "Rodríguez", "Pérez", "Fernández", "López",
"Martínez",
"García","Sánchez","Romero","Torres","Oviedo","Santillan","Escudero","Felice",
"Herrera"]
deportes = ["Fútbol", "Básquet", "Natación", "Tenis", "Atletismo", "Vóley",
"Hockey", "Handball","Rugby"]

#Con los datos base anteriores creamos una función para generar deportistas
utilizando 'random'.
def generar_deportistas(cantidad):
    deportistas = []
    for i in range(1, cantidad + 1):
        deportista = {
            "id": i,
            "dni": random.randint(10000000, 50000000),
            "nombre": random.choice(nombres),
            "apellido": random.choice(apellidos),
            "edad": random.randint(12, 60),
            "deporte": random.choice(deportes)
        }
        deportistas.append(deportista)
    return deportistas

# Definimos una funcion para la medición de tiempo utilizando la libreria 'time'.
def medir_tiempo(funcion, *args):
    inicio = time.time()
    resultado = funcion(*args)
    fin = time.time()
    return resultado, fin - inicio
```

```python
# Creamos 4 funciones de ordenamientos por 'TimSort' (combina MergeSort e
InsertionSort)
def ordenar_por_apellido(lista): return sorted(lista, key=lambda x:
x["apellido"])
def ordenar_por_edad(lista): return sorted(lista, key=lambda x: x["edad"])
def ordenar_por_deporte(lista): return sorted(lista, key=lambda x: x["deporte"])
def ordenar_por_dni(lista): return sorted(lista, key=lambda x: x["dni"])

#Creamos una funcion de Bubble Sort (edad).
def bubble_sort_por_edad(lista):
    lista = lista.copy()
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j]["edad"] > lista[j + 1]["edad"]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista

#Funcion de busqueda lineales por DNI:
def buscar_por_dni_lineal(lista, dni_buscado):
    for deportista in lista:
        if deportista["dni"] == dni_buscado:
            return deportista
    return None

#Funcion de busqueda binaria por DNI
def buscar_por_dni_binaria(lista_ordenada, dni_buscado):
    izquierda, derecha = 0, len(lista_ordenada) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista_ordenada[medio]["dni"] == dni_buscado:
            return lista_ordenada[medio]
        elif lista_ordenada[medio]["dni"] < dni_buscado:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return None

#Función de búsqueda lineal por deporte.
def buscar_por_deporte(lista, deporte_buscado):
    return [d for d in lista if d["deporte"].lower() == deporte_buscado.lower()]

# Función para mostrar en pantalla la lista generada aleatoriamente.
def mostrar_lista(lista):
    for d in lista:
        print(f"{d['id']:02d} | {d['dni']} | {d['nombre']} {d['apellido']} |
Edad: {d['edad']} | Deporte: {d['deporte']}")
```

```
# Función para crear el menú interactivo que se verá en pantalla.
# Primeramente ordenamos la lista por DNI, ya que lo requerimos para nuestra
búsqueda binaria; y es recomendable tenerla ordenada primeramente para no tener
que ordenarla cada vez que el usuario elige esa opción dentro del ciclo.

def menu(deportistas):
    deportistas_ordenados_dni = ordenar_por_dni(deportistas)
    continuar = True

    while continuar:
        print("\n=== MENÚ DEL SISTEMA DEPORTIVO ===")
        print("1. Mostrar lista original")
        print("2. Ordenar por apellido")
        print("3. Ordenar por edad ('TimSort')")
        print("4. Ordenar por edad ('BubbleSort')")
        print("5. Ordenar por deporte")
        print("6. Buscar por DNI (Lineal)")
        print("7. Buscar por DNI (Binaria)")
        print("8. Buscar por deporte")
        print("9. Salir")

        opcion = input("Seleccioná una opción: ")
        if opcion == "1":
            print("\n LISTA ORIGINAL")
            mostrar_lista(deportistas)

        elif opcion == "2":
            lista_ordenada, tiempo = medir_tiempo(ordenar_por_apellido,
deportistas)
            print("\n ORDENADA POR APELLIDO")
            mostrar_lista(lista_ordenada)
            print(f" Tiempo: {tiempo:.6f} segundos")

        elif opcion == "3":
            lista_ordenada, tiempo = medir_tiempo(ordenar_por_edad, deportistas)
            print("\n ORDENADA POR EDAD ('TimSort')")
            mostrar_lista(lista_ordenada)
            print(f" Tiempo: {tiempo:.6f} segundos")

        elif opcion == "4":
            lista_ordenada, tiempo = medir_tiempo(bubble_sort_por_edad,
deportistas)
            print("\n ORDENADA POR EDAD (BubbleSort)")
            mostrar_lista(lista_ordenada)
            print(f" Tiempo: {tiempo:.6f} segundos")

        elif opcion == "5":
            lista_ordenada, tiempo = medir_tiempo(ordenar_por_deporte,
deportistas)
            print("\n ORDENADA POR DEPORTE")
            mostrar_lista(lista_ordenada)
            print(f" Tiempo: {tiempo:.6f} segundos")
```

```
        elif opcion == "6":
            try:
                dni = int(input("Ingresá el DNI a buscar: "))
                resultado, tiempo = medir_tiempo(buscar_por_dni_lineal,
deportistas, dni)
                print("\n Resultado:" if resultado else " No encontrado.")
                if resultado: print(resultado)
                print(f" Tiempo: {tiempo:.6f} segundos")
            except ValueError:
                print(" DNI inválido.")

        elif opcion == "7":
            try:
                dni = int(input("Ingresá el DNI a buscar: "))
                resultado, tiempo = medir_tiempo(buscar_por_dni_binaria,
deportistas_ordenados_dni, dni)
                print("\n Resultado:" if resultado else " No encontrado.")
                if resultado: print(resultado)
                print(f" Tiempo: {tiempo:.6f} segundos")
            except ValueError:
                print(" DNI inválido.")

        elif opcion == "8":
            deporte = input("Ingresá el deporte a buscar: ")
            resultados, tiempo = medir_tiempo(buscar_por_deporte, deportistas,
deporte)
            if resultados:
                print(f"\n Deportistas en {deporte.title()}:")
                mostrar_lista(resultados)
            else:
                print(" No se encontraron deportistas con ese deporte.")
            print(f" Tiempo: {tiempo:.6f} segundos")



        elif opcion == "9":
            print(" Saliendo del sistema.")
            continuar = False
        else:
            print(" Opción inválida.")

# Ejecución de las funciones 'generar deportistas' y 'menú' que tendrán todo el
resto del proceso anidado.

deportistas = generar_deportistas(30)
menu(deportistas)
```

Decisiones de diseño:

Para resolver el problema de organización y consulta de datos en el sistema de gestión de deportistas, aplicamos distintos algoritmos de búsqueda y ordenamiento, priorizando la claridad, eficiencia y aplicabilidad.

Ordenamiento

Utilizamos la función sorted() de Python para ordenar listas por apellido, edad, deporte y DNI. Esta función aplica internamente el algoritmo **Timsort**, una combinación optimizada de **Merge Sort** e **Insertion Sort**, lo que garantiza eficiencia (O(n log n)) y estabilidad. Se eligió por su buen rendimiento en listas grandes y su facilidad de uso con el parámetro key.

Además, se implementó **Bubble Sort** de forma manual para ordenar por edad. Aunque es un algoritmo ineficiente ($O(n^2)$), se incluyó con fines didácticos para mostrar cómo funciona un ordenamiento paso a paso y permitir la comparación directa con sorted().

Búsqueda

Se aplicaron tres tipos de búsqueda:

- **Lineal**: útil para listas no ordenadas, se utilizó para buscar deportistas por DNI. Es fácil de programar, aunque poco eficiente en listas grandes.
- **Binaria**: requiere una lista previamente ordenada por DNI. Se incluyó para mostrar cómo una estructura ordenada permite mejorar la eficiencia (O(log n)) en búsquedas concretas.
- **Por deporte**: se utilizó una búsqueda por coincidencia textual para listar deportistas que practican una disciplina determinada. Esta búsqueda demuestra el uso de filtrado por condiciones específicas, útil en contextos reales.

En conjunto, estas decisiones nos permitieron construir un sistema flexible, funcional y representativo de los conceptos trabajados durante la cursada.

Validación del funcionamiento - Análisis de rendimiento y comparativa de algoritmos

Para validar el comportamiento de los algoritmos implementados, se realizaron pruebas de rendimiento con diferentes volúmenes de datos: 1.000, 10.000, 100.000 y 1.000.000 de personas (deportistas simulados). Se midió el tiempo de ejecución de los métodos de ordenamiento y búsqueda, expresado en segundos.

Metodo / Cant. Personas Segundos	1.000	10.000	100.000	1.000.000
Ordenamiento por 'TimeSort'	0,001006	0,004084	0,030478	0,183466
Ordenamiento por 'BubbleSort'	0,06384	5,301766	Crash	Crash
Busqueda DNI (lineal)	0	0,001931	0,00967	0,00944
Busqueda DNI (binaria)	0	0	0	0,000994

Las capturas de pantalla de la ejecución donde se obtuvieron estos tiempos se encuentra en el anexo.

Ordenamiento

- **Timsort (sorted() en Python):**
  Demostró ser altamente eficiente y escalable. El tiempo de ejecución creció de forma controlada a medida que aumentaba la cantidad de datos. Incluso con 1 millón de elementos, completó el ordenamiento en menos de 0,2 segundos.
- **Bubble Sort (manual):**
  Fue funcional con listas pequeñas (1.000), pero extremadamente ineficiente en volúmenes mayores. Con 10.000 elementos tardó más de 5 segundos, y al alcanzar 100.000 o más, directamente colapsó el sistema (Crash) por su complejidad $O(n^2)$, lo que confirma que no es adecuado para listas grandes.

Búsqueda por DNI

- **Lineal**:
  Funcionó correctamente en todos los casos, pero el tiempo aumentó con la cantidad de datos. Con 1 millón de registros, tardó casi 0,01 segundos, lo que puede parecer poco, pero demuestra que su rendimiento depende del tamaño total.
- **Binaria**:
  Fue la más eficiente. Su tiempo de búsqueda se mantuvo prácticamente constante y mínimo incluso con 1 millón de registros, gracias a su complejidad logarítmica O(log n).
  Es la opción ideal cuando la lista está previamente ordenada por DNI.

Conclusión de las pruebas

- Para ordenamiento, Timsort es claramente superior en rendimiento y estabilidad.
  Para búsqueda, la búsqueda binaria es significativamente más rápida que la lineal, siempre que se trabaje sobre una lista ordenada.

Estas pruebas no solo validan el correcto funcionamiento del sistema, sino que además permiten comparar objetivamente la eficiencia de los algoritmos según el contexto y volumen de datos, reforzando los conceptos estudiados durante la cursada.

Metodología Utilizada

El desarrollo del trabajo se llevó a cabo siguiendo una serie de etapas que permitieron integrar tanto los contenidos teóricos como la implementación práctica en Python.

Investigación previa

Se realizó una revisión de materiales del curso y fuentes complementarias para comprender los algoritmos de búsqueda y ordenamiento. Se consultaron apuntes de clase, documentación oficial de Python y recursos educativos como Python Docs y libros introductorios.

Diseño y pruebas del código

El sistema fue diseñado simulando una situación real: la gestión de datos de deportistas en un club. Primero se definió la estructura de los datos y las funcionalidades básicas. Luego, se implementaron las funciones de búsqueda y ordenamiento, incorporando diferentes algoritmos para comparar su rendimiento. Cada funcionalidad fue probada individualmente con distintos volúmenes de datos, utilizando mediciones de tiempo para observar el comportamiento de cada algoritmo. También se desarrolló un menú interactivo para facilitar el uso del sistema y simular una experiencia realista.

Herramientas utilizadas

- **Lenguaje de programación:** Python 3.11
- **IDE:** Visual Studio Code (VS Code)
- **Control de versiones:** Git y GitHub para respaldo y colaboración
- **Librerías:** Módulos estándar (random, time) para mantener el proyecto simple y portable

Trabajo colaborativo

El trabajo fue realizado en equipo por Astrid y Rodrigo, los cuales participamos activamente en todas las etapas del proyecto. La distribución de tareas fue complementaria:

- **Astrid** se enfocó en el análisis teórico de los algoritmos, la implementación del menú interactivo, y parte de las pruebas funcionales.
- **Rodrigo** se encargó del desarrollo de las funciones de búsqueda y ordenamiento, así como de la documentación técnica y la redacción inicial del marco teórico.

Ambos trabajaron de forma conjunta en la investigación previa, la revisión del código, la comparación de algoritmos y la producción del video explicativo.

Conclusión: Lo que aprendimos en el camino.

Este trabajo práctico nos permitió profundizar en el estudio y la implementación de algoritmos de búsqueda y ordenamiento, temas fundamentales dentro de la programación. A través del desarrollo de un caso práctico, pudimos pasar de la teoría a la práctica, enfrentándonos a decisiones concretas sobre qué algoritmos utilizar según el tipo y la estructura de los datos.

Uno de los principales aprendizajes fue comprender cómo diferentes algoritmos pueden impactar en la eficiencia y rendimiento de un programa. También aprendimos a evaluar cuál es el más adecuado según el contexto: por ejemplo, cuándo conviene usar una búsqueda lineal frente a una binaria, o cuándo un algoritmo de ordenamiento más simple (como bubble sort) puede ser útil a pesar de no ser el más eficiente.

Estos conocimientos tienen una gran utilidad tanto en proyectos académicos como profesionales. La búsqueda y el ordenamiento de información son tareas presentes en prácticamente cualquier aplicación real: desde motores de búsqueda, filtros en sistemas de usuarios, hasta la gestión de bases de datos o visualización de datos. Entender cómo funcionan estos algoritmos desde adentro nos prepara mejor para optimizar soluciones futuras.

Durante el desarrollo del trabajo surgieron algunas dificultades, como entender a fondo el funcionamiento de algoritmos recursivos o asegurar que ciertas búsquedas funcionaran correctamente solo sobre listas ordenadas. Estos desafíos fueron resueltos mediante pruebas, lectura de documentación adicional, y sobre todo mediante el trabajo colaborativo, donde cada integrante aportó ideas y soluciones.

Una de las mejoras que decidimos incorporar fue la inclusión de **visualizaciones gráficas** del funcionamiento de los algoritmos, lo que no solo facilitó nuestra comprensión, sino que también mejoró la presentación del trabajo. Como posibles extensiones futuras, podría explorarse el uso de estructuras de datos más complejas, o la implementación de algoritmos adicionales para análisis comparativos más profundos.

En conclusión, este trabajo no solo nos permitió aplicar conceptos clave, sino también desarrollar habilidades de análisis, diseño de soluciones y trabajo en equipo que resultan fundamentales en cualquier campo de la informática.

Bibliografía

- Python Software Foundation. (n.d.). *Python 3 documentation*. - https://docs.python.org/3/
- W3Schools. (n.d.). *Python How To - Sort a List*. https://www.w3schools.com/python/python_howto_sort.asp
- Mise Olivera, A. (2020). *Algoritmos de Búsqueda y Ordenamiento* [Video]. YouTube. https://www.youtube.com/watch?v=haF4P8kF4Ik
- Mise Olivera, A. (2020). *Búsqueda binaria y búsqueda secuencial en Python* [Video]. YouTube. https://www.youtube.com/watch?v=u1QuRbx-_x4
- Material de clase de la cátedra *Programación I*. (2025). Universidad Tecnológica Nacional.

Anexos

En este anexo se adjuntan capturas del programa funcionando, mostrando cada parte en funcionamiento.

Menú iterativo:

```
=== MENÚ DEL SISTEMA DEPORTIVO ===
1. Mostrar lista original
2. Ordenar por apellido
3. Ordenar por edad ('TimSort')
4. Ordenar por edad ('BubbleSort')
5. Ordenar por deporte
6. Buscar por DNI (Lineal)
7. Buscar por DNI (Binaria)
8. Buscar por deporte
9. Salir
Seleccioná una opción:
```

Las siguientes **pruebas visuales** se realizan con una **lista de** 30 deportistas, generada aleatoriamente.

Opción 1 - Mostrar lista original.

```
Seleccioná una opción: 1

 LISTA ORIGINAL
01 | 12175014 | Agustín Herrera | Edad: 45 | Deporte: Básquet
02 | 28770205 | Agustín Herrera | Edad: 45 | Deporte: Fútbol
03 | 34558420 | Jaime Fernández | Edad: 47 | Deporte: Hockey
04 | 48543330 | María Santillan | Edad: 50 | Deporte: Vóley
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
06 | 10644538 | María Rodríguez | Edad: 14 | Deporte: Atletismo
07 | 40680012 | Esteban Romero | Edad: 52 | Deporte: Tenis
08 | 48703589 | Esteban Fernández | Edad: 27 | Deporte: Hockey
09 | 33637032 | Valentina García | Edad: 38 | Deporte: Handball
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
11 | 48100387 | Jaime Felice | Edad: 36 | Deporte: Vóley
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
13 | 14435123 | Juan Torres | Edad: 31 | Deporte: Básquet
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
18 | 48260817 | Sandra Oviedo | Edad: 59 | Deporte: Vóley
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
20 | 27118426 | Lucas Fernández | Edad: 16 | Deporte: Atletismo
21 | 47483701 | Agustín Romero | Edad: 49 | Deporte: Tenis
22 | 33470625 | Lucas Herrera | Edad: 19 | Deporte: Básquet
23 | 31619141 | Jaime Escudero | Edad: 39 | Deporte: Vóley
24 | 20892431 | Jaime Martínez | Edad: 23 | Deporte: Fútbol
25 | 18869724 | Tomás Torres | Edad: 52 | Deporte: Vóley
26 | 37601999 | Pedro Escudero | Edad: 28 | Deporte: Básquet
27 | 42224078 | María Fernández | Edad: 18 | Deporte: Atletismo
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
29 | 43415063 | Benjamín Martínez | Edad: 50 | Deporte: Vóley
30 | 12374352 | Esteban Oviedo | Edad: 44 | Deporte: Hockey
```

Como se puede observar, se generó una lista aleatoria de 30 personas, combinando Nombre, Apellido, Edades (rango 12-60) y deportes.

Opción 2: Ordenar por apellido.

```
 ORDENADA POR APELLIDO
23 | 31619141 | Jaime Escudero | Edad: 39 | Deporte: Vóley
26 | 37601999 | Pedro Escudero | Edad: 28 | Deporte: Básquet
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
11 | 48100387 | Jaime Felice | Edad: 36 | Deporte: Vóley
03 | 34558420 | Jaime Fernández | Edad: 47 | Deporte: Hockey
08 | 48703589 | Esteban Fernández | Edad: 27 | Deporte: Hockey
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
20 | 27118426 | Lucas Fernández | Edad: 16 | Deporte: Atletismo
27 | 42224078 | María Fernández | Edad: 18 | Deporte: Atletismo
09 | 33637032 | Valentina García | Edad: 38 | Deporte: Handball
01 | 12175014 | Agustín Herrera | Edad: 45 | Deporte: Básquet
02 | 28770205 | Agustín Herrera | Edad: 45 | Deporte: Fútbol
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
22 | 33470625 | Lucas Herrera | Edad: 19 | Deporte: Básquet
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
24 | 20892431 | Jaime Martínez | Edad: 23 | Deporte: Fútbol
29 | 43415063 | Benjamín Martínez | Edad: 50 | Deporte: Vóley
18 | 48260817 | Sandra Oviedo | Edad: 59 | Deporte: Vóley
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
30 | 12374352 | Esteban Oviedo | Edad: 44 | Deporte: Hockey
06 | 10644538 | María Rodríguez | Edad: 14 | Deporte: Atletismo
07 | 40680012 | Esteban Romero | Edad: 52 | Deporte: Tenis
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
21 | 47483701 | Agustín Romero | Edad: 49 | Deporte: Tenis
04 | 48543330 | María Santillan | Edad: 50 | Deporte: Vóley
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
13 | 14435123 | Juan Torres | Edad: 31 | Deporte: Básquet
25 | 18869724 | Tomás Torres | Edad: 52 | Deporte: Vóley
```

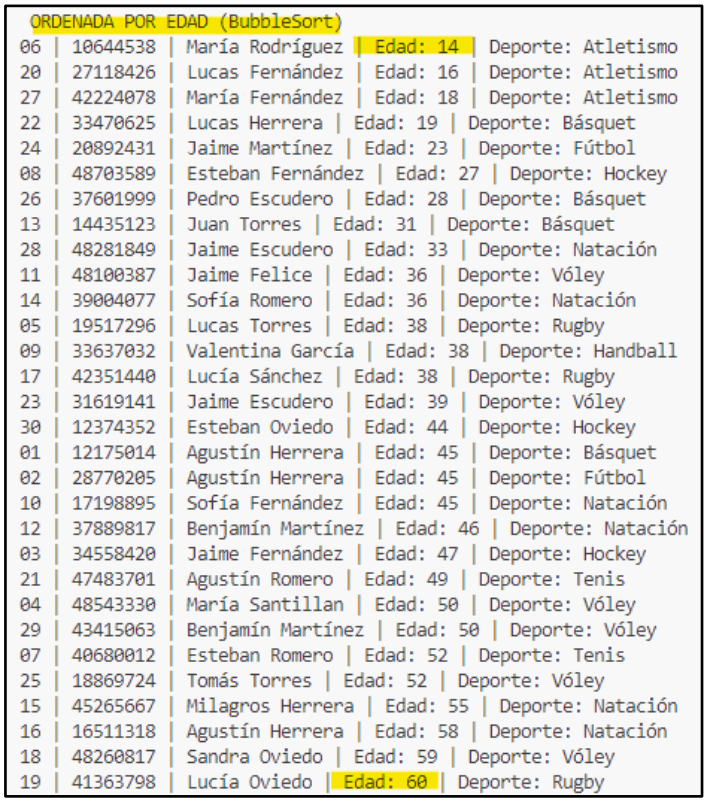
Los apellidos fueron ordenados alfabéticamente, comenzando por Escudero y finalizando con el apellido Torres.

Opción 3: Ordenar por edad (‘TimeSort’)

```
 ORDENADA POR EDAD ('TimSort')
06 | 10644538 | María Rodríguez | Edad: 14 | Deporte: Atletismo
20 | 27118426 | Lucas Fernández | Edad: 16 | Deporte: Atletismo
27 | 42224078 | María Fernández | Edad: 18 | Deporte: Atletismo
22 | 33470625 | Lucas Herrera | Edad: 19 | Deporte: Básquet
24 | 20892431 | Jaime Martínez | Edad: 23 | Deporte: Fútbol
08 | 48703589 | Esteban Fernández | Edad: 27 | Deporte: Hockey
26 | 37601999 | Pedro Escudero | Edad: 28 | Deporte: Básquet
13 | 14435123 | Juan Torres | Edad: 31 | Deporte: Básquet
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
11 | 48100387 | Jaime Felice | Edad: 36 | Deporte: Vóley
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
09 | 33637032 | Valentina García | Edad: 38 | Deporte: Handball
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
23 | 31619141 | Jaime Escudero | Edad: 39 | Deporte: Vóley
30 | 12374352 | Esteban Oviedo | Edad: 44 | Deporte: Hockey
01 | 12175014 | Agustín Herrera | Edad: 45 | Deporte: Básquet
02 | 28770205 | Agustín Herrera | Edad: 45 | Deporte: Fútbol
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
03 | 34558420 | Jaime Fernández | Edad: 47 | Deporte: Hockey
21 | 47483701 | Agustín Romero | Edad: 49 | Deporte: Tenis
04 | 48543330 | María Santillan | Edad: 50 | Deporte: Vóley
29 | 43415063 | Benjamín Martínez | Edad: 50 | Deporte: Vóley
07 | 40680012 | Esteban Romero | Edad: 52 | Deporte: Tenis
25 | 18869724 | Tomás Torres | Edad: 52 | Deporte: Vóley
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
18 | 48260817 | Sandra Oviedo | Edad: 59 | Deporte: Vóley
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
```

Las edades fueron ordenadas desde la persona con menor edad ‘14 años’ hasta la mayor persona de ‘60 años’.

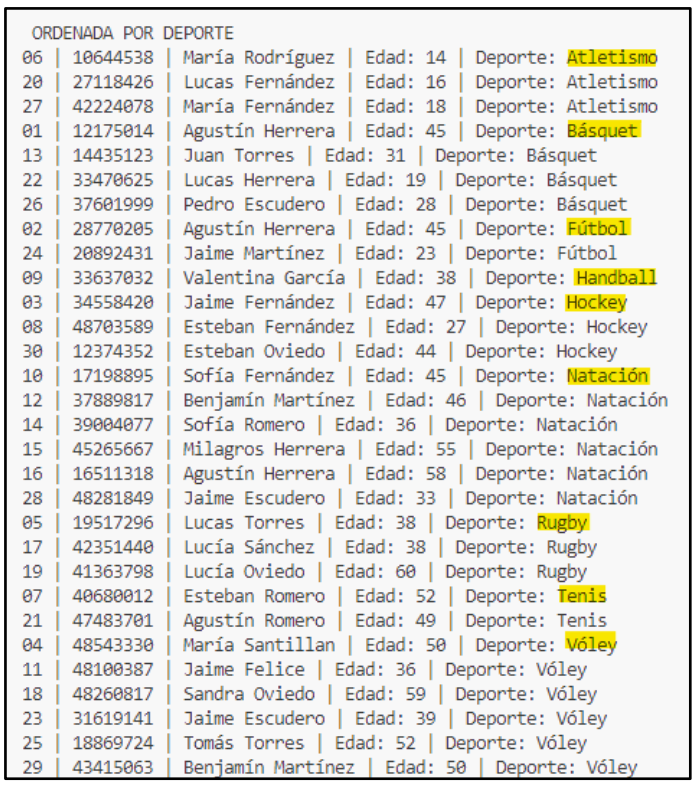
Opción 4: Ordenar por edad (‘BubbleSort’)



```
 ORDENADA POR EDAD (BubbleSort)
06 | 10644538 | María Rodríguez | Edad: 14 | Deporte: Atletismo
20 | 27118426 | Lucas Fernández | Edad: 16 | Deporte: Atletismo
27 | 42224078 | María Fernández | Edad: 18 | Deporte: Atletismo
22 | 33470625 | Lucas Herrera | Edad: 19 | Deporte: Básquet
24 | 20892431 | Jaime Martínez | Edad: 23 | Deporte: Fútbol
08 | 48703589 | Esteban Fernández | Edad: 27 | Deporte: Hockey
26 | 37601999 | Pedro Escudero | Edad: 28 | Deporte: Básquet
13 | 14435123 | Juan Torres | Edad: 31 | Deporte: Básquet
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
11 | 48100387 | Jaime Felice | Edad: 36 | Deporte: Vóley
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
09 | 33637032 | Valentina García | Edad: 38 | Deporte: Handball
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
23 | 31619141 | Jaime Escudero | Edad: 39 | Deporte: Vóley
30 | 12374352 | Esteban Oviedo | Edad: 44 | Deporte: Hockey
01 | 12175014 | Agustín Herrera | Edad: 45 | Deporte: Básquet
02 | 28770205 | Agustín Herrera | Edad: 45 | Deporte: Fútbol
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
03 | 34558420 | Jaime Fernández | Edad: 47 | Deporte: Hockey
21 | 47483701 | Agustín Romero | Edad: 49 | Deporte: Tenis
04 | 48543330 | María Santillan | Edad: 50 | Deporte: Vóley
29 | 43415063 | Benjamín Martínez | Edad: 50 | Deporte: Vóley
07 | 40680012 | Esteban Romero | Edad: 52 | Deporte: Tenis
25 | 18869724 | Tomás Torres | Edad: 52 | Deporte: Vóley
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
18 | 48260817 | Sandra Oviedo | Edad: 59 | Deporte: Vóley
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
```

Al igual que el ejemplo anterior, se ordenan según la edad, de menor a mayor pero utilizando otro algoritmo de ordenamiento. **En próximos pasos compararemos los tiempos de ordenamiento de cada método.**

Opción 5: Ordenar por deporte



```
 ORDENADA POR DEPORTE
06 | 10644538 | María Rodríguez | Edad: 14 | Deporte: Atletismo
20 | 27118426 | Lucas Fernández | Edad: 16 | Deporte: Atletismo
27 | 42224078 | María Fernández | Edad: 18 | Deporte: Atletismo
01 | 12175014 | Agustín Herrera | Edad: 45 | Deporte: Básquet
13 | 14435123 | Juan Torres | Edad: 31 | Deporte: Básquet
22 | 33470625 | Lucas Herrera | Edad: 19 | Deporte: Básquet
26 | 37601999 | Pedro Escudero | Edad: 28 | Deporte: Básquet
02 | 28770205 | Agustín Herrera | Edad: 45 | Deporte: Fútbol
24 | 20892431 | Jaime Martínez | Edad: 23 | Deporte: Fútbol
09 | 33637032 | Valentina García | Edad: 38 | Deporte: Handball
03 | 34558420 | Jaime Fernández | Edad: 47 | Deporte: Hockey
08 | 48703589 | Esteban Fernández | Edad: 27 | Deporte: Hockey
30 | 12374352 | Esteban Oviedo | Edad: 44 | Deporte: Hockey
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
07 | 40680012 | Esteban Romero | Edad: 52 | Deporte: Tenis
21 | 47483701 | Agustín Romero | Edad: 49 | Deporte: Tenis
04 | 48543330 | María Santillan | Edad: 50 | Deporte: Vóley
11 | 48100387 | Jaime Felice | Edad: 36 | Deporte: Vóley
18 | 48260817 | Sandra Oviedo | Edad: 59 | Deporte: Vóley
23 | 31619141 | Jaime Escudero | Edad: 39 | Deporte: Vóley
25 | 18869724 | Tomás Torres | Edad: 52 | Deporte: Vóley
29 | 43415063 | Benjamín Martínez | Edad: 50 | Deporte: Vóley
```

Las personas fueron ordenadas y agrupadas por el deporte que practican, desde ‘Atletismo’ hasta ‘Voley’.

Opción 6: Búsqueda por DNI (Lineal)

```
Seleccioná una opción: 6
Ingresá el DNI a buscar: 34558420

 Resultado:
{'id': 3, 'dni': 34558420, 'nombre': 'Jaime', 'apellido': 'Fernández', 'edad': 47, 'deporte': 'Hockey'}
 Tiempo: 0.000000 segundos
```

```
 Seleccioná una opción: 6
 Ingresá el DNI a buscar: 39850315
 ERROR: No encontrado.
  Tiempo: 0.000000 segundos
```

Ejemplo realizado en una búsqueda lineal de DNI entre 30 personas para mostrar el funcionamiento, luego se realizará en una base más amplia para comparación de tiempos.
La segunda captura indica el error cuando no se encuentra el DNI indicado.

Opción 7: Búsqueda por DNI (Binaria)

```
 Seleccioná una opción: 7
 Ingresá el DNI a buscar: 19517296

  Resultado:
{'id': 5, 'dni': 19517296, 'nombre': 'Lucas', 'apellido': 'Torres', 'edad': 38, 'deporte': 'Rugby'}
  Tiempo: 0.000000 segundos
```

```
 Seleccioná una opción: 7
 Ingresá el DNI a buscar: 39850315
  No encontrado.
  Tiempo: 0.000000 segundos
```

Ejemplo realizado en una búsqueda binaria de DNI entre 30 personas para mostrar el funcionamiento, luego se realizará en una base más amplia para comparación de tiempos.
La segunda captura indica el error cuando no se encuentra el DNI indicado.

Opción 8: Búsqueda por deporte

```
Seleccioná una opción: 8
Ingresá el deporte a buscar: rugby

 Deportistas en Rugby:
05 | 19517296 | Lucas Torres | Edad: 38 | Deporte: Rugby
17 | 42351440 | Lucía Sánchez | Edad: 38 | Deporte: Rugby
19 | 41363798 | Lucía Oviedo | Edad: 60 | Deporte: Rugby
 Tiempo: 0.000000 segundos
```

```
Seleccioná una opción: 8
Ingresá el deporte a buscar: natación

 Deportistas en Natación:
10 | 17198895 | Sofía Fernández | Edad: 45 | Deporte: Natación
12 | 37889817 | Benjamín Martínez | Edad: 46 | Deporte: Natación
14 | 39004077 | Sofía Romero | Edad: 36 | Deporte: Natación
15 | 45265667 | Milagros Herrera | Edad: 55 | Deporte: Natación
16 | 16511318 | Agustín Herrera | Edad: 58 | Deporte: Natación
28 | 48281849 | Jaime Escudero | Edad: 33 | Deporte: Natación
 Tiempo: 0.000000 segundos
```

El sistema permite realizar una búsqueda ordenada por disciplina deportiva, como se puede apreciar podemos buscar / filtrar los deportistas por cada disciplina.

Pruebas comparativas:

Para las pruebas comparativas se ha modificado la base de 30 personas a diferentes escalas (1000 - 10.000 - 100.000) personas.

Pruebas de tiempos por 1000 personas:

Ordenados por edad por 'TimeSort'

```
 867 | 48734496 | Mateo Torres | Edad: 60 | Deporte: Handball
 935 | 17411415 | Ezequiel Fernández | Edad: 60 | Deporte: Handball
 958 | 29412285 | Sandra Martínez | Edad: 60 | Deporte: Básquet
 990 | 49430644 | Pedro Torres | Edad: 60 | Deporte: Fútbol
 Tiempo: 0.001006 segundos
```

Ordenados por edad por 'BubbleSort'

```
867 | 48734496 | Mateo Torres | Edad: 60 | Deporte: Handball
935 | 17411415 | Ezequiel Fernández | Edad: 60 | Deporte: Handball
958 | 29412285 | Sandra Martínez | Edad: 60 | Deporte: Básquet
990 | 49430644 | Pedro Torres | Edad: 60 | Deporte: Fútbol
 Tiempo: 0.063840 segundos
```

Búsqueda de DNI (Lineal)

```
 Seleccioná una opción: 6
 Ingresá el DNI a buscar: 43683735

  Resultado:
 {'id': 971, 'dni': 43683735, 'nombre': 'Lucía', 'apellido': 'Romero', 'edad': 46, 'deporte': 'Vóley'}
  Tiempo: 0.000000 segundos
```

Búsqueda de DNI (Binaria)

```
 Seleccioná una opción: 7
 Ingresá el DNI a buscar: 28921323

  Resultado:
 {'id': 994, 'dni': 28921323, 'nombre': 'Mateo', 'apellido': 'Pérez', 'edad': 24, 'deporte': 'Tenis'}
  Tiempo: 0.000000 segundos
```

Pruebas de tiempo por 10.000 personas:

Ordenados por edad por 'TimeSort'

```
 9768 | 32220446 | Tomás Felice | Edad: 60 | Deporte: Vóley
 9837 | 34243390 | Pedro Escudero | Edad: 60 | Deporte: Vóley
 9915 | 41254113 | Camila Martínez | Edad: 60 | Deporte: Rugby
 9938 | 43669190 | Benjamín López | Edad: 60 | Deporte: Natación
  Tiempo: 0.004084 segundos
```

Ordenados por edad por 'BubbleSort'

```
 9768 | 32220446 | Tomás Felice | Edad: 60 | Deporte: Vóley
 9837 | 34243390 | Pedro Escudero | Edad: 60 | Deporte: Vóley
 9915 | 41254113 | Camila Martínez | Edad: 60 | Deporte: Rugby
 9938 | 43669190 | Benjamín López | Edad: 60 | Deporte: Natación
  Tiempo: 5.301766 segundos
```

Búsqueda de DNI (Lineal)

```
Seleccioná una opción: 6
Ingresá el DNI a buscar: 12122541

 Resultado:
{'id': 8882, 'dni': 12122541, 'nombre': 'María', 'apellido': 'Herrera', 'edad': 60, 'deporte': 'Fútbol'}
 Tiempo: 0.001931 segundos
```

Búsqueda de DNI (Binaria)

```
Seleccioná una opción: 7
Ingresá el DNI a buscar: 12122541

 Resultado:
{'id': 8882, 'dni': 12122541, 'nombre': 'María', 'apellido': 'Herrera', 'edad': 60, 'deporte': 'Fútbol'}
 Tiempo: 0.000000 segundos
```

Pruebas de tiempo por 100.000 personas:

Ordenados por edad por 'TimeSort'

```
99839 | 48134484 | Benjamín Oviedo | Edad: 60 | Deporte: Básquet
99864 | 43299306 | Pedro Torres | Edad: 60 | Deporte: Hockey
99888 | 32028825 | Benjamín García | Edad: 60 | Deporte: Hockey
99945 | 12273352 | Ezequiel Torres | Edad: 60 | Deporte: Básquet
 Tiempo: 0.030478 segundos
```

Ordenados por edad por 'BubbleSort'

Más de 10 minutos procesando y no se ejecutó. - **Crash**.

Búsqueda de DNI (Lineal)

```
Seleccioná una opción: 6
Ingresá el DNI a buscar: 23894309

 Resultado:
{'id': 98997, 'dni': 23894309, 'nombre': 'Tomás', 'apellido': 'Santillan', 'edad': 20, 'deporte': 'Rugby'}
 Tiempo: 0.009670 segundos
```

Búsqueda de DNI (Binaria)

```
Seleccioná una opción: 7
Ingresá el DNI a buscar: 39516570

 Resultado:
{'id': 98996, 'dni': 39516570, 'nombre': 'Benjamín', 'apellido': 'Felice', 'edad': 59, 'deporte': 'Básquet'}
 Tiempo: 0.000000 segundos
```

Pruebas de tiempo por 1.000.000 personas:

Ordenados por edad por 'TimeSort'

```
999827 | 31144610 | Juan Herrera | Edad: 60 | Deporte: Tenis
999877 | 49588565 | Milagros Oviedo | Edad: 60 | Deporte: Atletismo
999905 | 39277526 | Tomás Pérez | Edad: 60 | Deporte: Vóley
999969 | 44634588 | Valentina Martínez | Edad: 60 | Deporte: Fútbol
 Tiempo: 0.183466 segundos
```

Ordenados por edad por 'BubbleSort'

Más de 10 minutos procesando y no se ejecutó. - **Crash**.

Búsqueda de DNI (Lineal)

```
Seleccioná una opción: 6
Ingresá el DNI a buscar: 39277525

 Resultado:
{'id': 466819, 'dni': 39277525, 'nombre': 'Sandra', 'apellido': 'López', 'edad': 28, 'deporte': 'Hockey'}
 Tiempo: 0.038075 segundos
```

Búsqueda de DNI (Binaria)

```
Seleccioná una opción: 7
Ingresá el DNI a buscar: 39277526
 No encontrado.
 Tiempo: 0.000994 segundos
```