

## Trabajo Práctico INTEGRADOR N° 2

# ALGORITMOS DE BÚSQUEDA Y ORDENAMIENTO EN PYTHON

—

## APLICACIÓN: BÚSQUEDA DE ABOGADOS MATRICULADOS SEGÚN SUS ESPECIALIDADES Y OTROS FILTROS ÚTILES APLICABLES

### Alumnos

- Gaston Alberto Cejas – [gaston.cejas@gmail.com](mailto:gaston.cejas@gmail.com)
- Diego Matias Carrizo, [diegocarrizo85@gmail.com](mailto:diegocarrizo85@gmail.com)

### Materia

Programación I – Grupo 11,

### Profesor

Ariel Enferrel

### Tutora:

Martina Zabala

### Fecha de entrega

09 de junio de 2025

## ÍNDICE

1.	<b>Introducción .....</b>	p. 3
2.	<b>Marco Teórico.....</b>	p.5
3.	<b>Caso Práctico.....</b>	p. 9
4.	<b>Metodología Utilizada.....</b>	p. 15
5.	<b>Resultados Obtenidos.....</b>	p. 16
6.	<b>Conclusiones.....</b>	p. 19
7.	<b>Bibliografía.....</b>	p. 21
8.	<b>Anexo.....</b>	p. 22

# INTRODUCCIÓN

En la programación, los algoritmos de búsqueda y ordenamiento son fundamentales para la gestión eficiente de la información. Comprender cómo se implementan y en qué situaciones se aplican es clave para el desarrollo de software efectivo. Este trabajo explora estos algoritmos.

El tema se eligió debido a que es necesario para resolver tanto tareas simples del día a día, como tareas complejas en base de datos o listas muy grandes. En este trabajo se demuestra con un ejemplo como una simple búsqueda lineal de un número en una lista puede requerir una cantidad de tiempo que se incrementa de manera lineal a medida que se aumenta el tamaño de la lista de elementos.

Los algoritmos de búsqueda y ordenamiento son fundamentales en la programación por las siguientes razones:

## **-Eficiencia en el manejo de datos:**

- Muchas aplicaciones trabajan con grandes volúmenes de datos (bases de datos, redes sociales, sistemas de recomendación, etc.).
- Un algoritmo eficiente de búsqueda u ordenamiento reduce el tiempo de procesamiento y mejora la experiencia del usuario. Ejemplo: Buscar un nombre en una lista de millones de registros es mucho más rápido con un algoritmo de búsqueda binaria (si los datos están ordenados) que con una búsqueda lineal.

## **-Optimización de recursos:**

- Algoritmos como QuickSort, MergeSort o Binary Search están diseñados para minimizar el uso de memoria y CPU.
- En sistemas críticos (como transacciones bancarias o motores de búsqueda), incluso una mejora de milisegundos es crucial.

## **-Base para problemas complejos:**

- Muchos problemas avanzados (grafos, inteligencia artificial, procesamiento de imágenes) dependen de operaciones de búsqueda y ordenamiento.

Ejemplo: Un sistema de GPS (como Google Maps) usa algoritmos como Dijkstra (algoritmo que encuentra la ruta más corta desde un nodo origen a todos los demás nodos en un grafo ponderado), que requieren estructuras de datos ordenadas para encontrar rutas óptimas.

**-Preparación para entrevistas técnicas:**

- Las empresas evalúan el conocimiento de estos algoritmos porque demuestran lógica y capacidad para resolver problemas eficientemente. Preguntas como "Ordena una lista en tiempo  $O(n \log n)$ " o "Encuentra un elemento en un arreglo rotado" son comunes.

**-Fundamento teórico:**

Estos algoritmos enseñan conceptos clave de la ciencia de la computación:

- ✓ Complejidad algorítmica (Big-O notation:  $O(1)$ ,  $O(n)$ ,  $O(\log n)$ , etc.).
- ✓ Recursividad (ejemplo: MergeSort).
- ✓ Divide y vencerás (QuickSort, Binary Search).

**Ejemplos Prácticos:**

- ✓ Búsqueda Binaria: Usada en sistemas de índices de bases de datos.
- ✓ QuickSort: Implementado en librerías estándar (como `sort()` en Python).
- ✓ Counting Sort: Útil para ordenar datos con rangos limitados (ejemplo: edades de usuarios).

**Conclusión:**

Dominar estos algoritmos no solo mejora el rendimiento de tus programas, sino que también te ayuda a pensar de manera estructurada al enfrentar problemas nuevos. Son la base para construir soluciones escalables y eficientes.

**Objetivo:**

El principal objetivo de este trabajo es aplicar y utilizar los algoritmos más importantes de búsqueda y ordenamiento de datos para resolver un problema de la vida cotidiana.

## 2. Marco teórico

Los algoritmos de ordenamiento permiten reorganizar una colección de elementos de acuerdo con un criterio específico como ser: de mayor a menor, de menor a mayor, alfabéticamente, etc.

Entre los más comunes se encuentran:

- **Bubble Sort**, algoritmo de búsqueda de burbuja: compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto.

- **Insertion Sort**: construye una lista ordenada tomando elementos uno a uno e insertándolos en la posición adecuada.

- **Selection Sort**: selecciona el elemento más pequeño del arreglo y lo coloca en su posición final.

Por otro lado, los algoritmos de búsqueda tienen como objetivo encontrar un valor dentro de una colección.

Destacamos:

- **Búsqueda lineal**: recorre secuencialmente todos los elementos hasta encontrar el deseado. Si bien es el método más fácil de entender y es la primera opción que se nos viene a la cabeza, tiene la desventaja de que en el peor de los casos debe recorrer la lista completa para encontrar el objetivo. (ver archivo Python adjunto donde se registra el tiempo de ejecución de una búsqueda)

- **Búsqueda binaria**: eficiente en listas ordenadas, divide el espacio de búsqueda en mitades sucesivas.

La búsqueda binaria funciona de la siguiente manera:

1. Se compara el elemento central con el valor buscado.
2. Si son iguales, se retorna la posición.
3. Si el valor es menor, se repite la búsqueda en la mitad izquierda.
4. Si es mayor, en la mitad derecha.

A continuación, vamos a presentar las funciones en Python desarrolladas para realizar la demostración en el lenguaje de programación elegido:

## # Algoritmos de búsqueda

### #función en python de búsqueda lineal usada para el trabajo

```
def busqueda_lineal(lista, elemento):  
    comparaciones = 0  
    tiempo_inicial = time.time() #registro del tiempo inicial  
    for i in range(len(lista)):  
        comparaciones += 1 #contador de comparaciones  
        if lista[i] == elemento: #comparacion elemento a elemento  
            tiempo_final = time.time() #registro del tiempo final  
            tiempo_employado = tiempo_final-tiempo_inicial #calculo tiempo  
            return i, comparaciones, tiempo_employado  
    tiempo_final = time.time()  
    tiempo_employado = tiempo_final-tiempo_inicial #tiempo empleado  
    return -1, comparaciones, tiempo_employado
```

### #función en python de búsqueda binaria usada para el trabajo

```
def busqueda_binaria(lista, elemento):  
    comparaciones = 0 #contador de comparaciones  
    izquierda, derecha = 0, len(lista) - 1 #inicializo los extremos de busqueda  
    tiempo_inicial = time.time() #registro tiempo  
    while izquierda <= derecha: #condicion de ciclo  
        comparaciones += 1 #incremento comparaciones  
        medio = (izquierda + derecha) // 2 #calculo la posicion media de la lista  
        if lista[medio] == elemento: #si el medio coincide con la busqueda
```

```
        tiempo_final = time.time()

        tiempo_empleado=tiempo_final-tiempo_inicial#tiempo empleado

        return medio, comparaciones, tiempo_empleado

    elif lista[medio] < elemento: #caso que el valor del medio sea menor

        izquierda = medio + 1 #modifico el extremo izquierdo del rango

    else:

        derecha = medio - 1 #modifico el extremo derecho del rango

    tiempo_final = time.time()

    tiempo_empleado = tiempo_final-tiempo_inicial #calculo el tiempo empleado

    return -1, comparaciones,tiempo_empleado
```

## # Algoritmos de ordenamiento

### #función en python de ordenamiento de burbuja usada para el trabajo

```
def ordenamiento_de_burbuja(lista):

    n = len(lista) #obtengo el tamaño de la lista

    operaciones = 0 #inicializo contador

    print('Lista original', lista)

    tiempo_inicial = time.time()#registra el tiempo inicial de la operacion

    for i in range(n):

        for j in range(0, n-i-1):

            operaciones += 1

            if lista[j] > lista[j+1]: #si el primero es mayor al adyacente

                lista[j], lista[j+1] = lista[j+1], lista[j] #intercambia posicion

    print(f"Paso {i+1}: {lista}")#muestro avance de ordenamiento

    tiempo_final = time.time()

    tiempo_empleado = tiempo_final-tiempo_inicial #calculo el tiempo empleado

    return lista, operaciones, tiempo_empleado
```

### **#función en python de ordenamiento por selección usada para el trabajo**

```
def ordenamiento_por_seleccion(lista):  
    operaciones = 0  
    tiempo_inicial = time.time() #registra el tiempo inicial de la operacion  
    for i in range(len(lista)):#recorre la lista en toda su longitud  
        minimo_posicion = i #toma la posicion minimo  
        for j in range(i+1, len(lista)):#recorre el resto de la lista  
            operaciones += 1 #incrementa contador  
            if lista[minimo_posicion] > lista[j]: #el valor es realmente menor  
                minimo_posicion = j #reasigna la posicion  
    lista[i], lista[minimo_posicion] = lista[minimo_posicion], lista[i] #para luego cambiarla al  
    lugar correcto  
    print(f"Paso {i+1}: {lista}")#muestro progreso  
    tiempo_final = time.time()  
    tiempo_empleado = tiempo_final-tiempo_inicial #calculo tiempo empleado  
    return lista, operaciones, tiempo_empleado
```

### **#función en python para crear una lista de números aleatoria usada para el trabajo**

```
def crea_lista_numeros_aleatorios(tamano):  
    #los numeros no se repiten  
    lista = random.sample(range(1, tamano+1), tamano) # Lista generada  
    return lista
```



### 3. Caso práctico: Búsqueda de abogados

Partimos de la necesidad, en el ámbito de la abogacía, de un cliente a la hora de elegir a un letrado. Generalmente la persona común sólo sabe que necesita a un abogado pero no tiene el conocimiento para detallar el área específica de su necesidad.

Proponemos entonces un programa que contiene una lista de abogados en el que están detallados varios items, a saber: nombre, numero matricula, años de experiencia, volumen de casos en cada área, cantidad de postgrados, precios por consulta, datos de contacto y una cantidad de detalles más que pueden implementarse a posterior.

Con esto el cliente puede ir determinando criterios de búsqueda para llegar a encontrar el letrado apropiado para su problema.

**La descripción del código propuesto es el siguiente:**

#### Líneas 1–2: Importación de bibliotecas

```
import pandas as pd
```

```
import time
```

- pandas: librería de Python especializada en manejar tablas de datos (DataFrames). Aquí la usamos para leer el Excel y filtrar filas.
- time: módulo de funciones de tiempo. Se importó originalmente para medir la duración de algoritmos, pero en esta versión no lo usamos activamente (podría eliminarse).

**IMPORTANTE:** Para poder utilizar dicha librería, fue necesaria la instalación instalar en nuestro entorno de Python **los paquetes pandas** (para gestionar y manipular datos en forma de tablas) **y openpyxl** (como motor que permite a pandas leer y escribir archivos Excel .xlsx), antes de lanzar el programa, ya que sin ellos, no se podría haber realizado la lectura del documento utilizado.

### Líneas 5–13: Configuración y carga de datos:

```
# ----- Configuración y carga de datos -----  
DATA_FILE = r"C:\Users\Usuario\Desktop\Gaston\Programacion UTN\UTN-TUPaD-P1---Fork\10 Búsqueda y ordenamier  
  
def cargar_datos():  
    try:  
        df = pd.read_excel(DATA_FILE, engine="openpyxl")  
    except FileNotFoundError:  
        print(f"Error: No se encontró el archivo '{DATA_FILE}'.")  
        exit()  
    return df
```

✓ **DATA\_FILE = r"...ruta...xlsx"**

o Aquí definimos una variable que llama DATA\_FILE y contiene la dirección donde está el archivo de Excel con la lista de abogados.

o La r delante de las comillas indica “string crudo”, que ayuda a no confundir las barras de Windows (\) con otras cosas.

✓ **Función cargar\_datos()**

o Busca el archivo, lo abre y lo convierte en una “tabla” dentro de la memoria.

o try ... except FileNotFoundError: Intentamos abrir el archivo; si no existe, en lugar de romperse todo, capturamos ese error y mostramos un mensaje amigable (“Error: No se encontró el archivo...”) y luego detenemos el programa.

o return df: Si todo salió bien, devolvemos (entregamos) esa tabla de datos para usarla después.

### Líneas 16–24: Definición de especialidades

```
16  
17 # Paso 1: Definir especialidades disponibles  
18 especialidades = [  
19     "Derecho Civil", "Amparos de Salud", "Nuevas tecnologías", "Derecho Penal",  
20     "Derecho Tributario", "Derecho de Familia", "Derecho Laboral",  
21     "Derecho Internacional Público", "Derecho Internacional Privado",  
22     "Derecho Constitucional", "Derecho Administrativo", "Derecho Comercial",  
23     "Defensa del Consumidor", "Derechos de incidencia colectiva",  
24     "Derecho aduanero", "Derecho previsional"  
25 ]
```

Aquí simplemente tenemos un arreglo de texto con 16 nombres de áreas del Derecho (Civil, Penal, Laboral, etc.).

Es como si hiciéramos un menú con todas las posibles “especialidades” para luego preguntarle

al usuario: "¿Cuál elegís?". "Derecho Civil", ..., "Derecho previsional". Es simplemente una lista de las 16 áreas del derecho que luego se muestran al usuario para filtrar.

## Algoritmos de búsqueda y ordenamiento

### Líneas 26–69

- `busqueda_lineal` (27–32)
- `busqueda_binaria` (35–46)
- `ordenamiento_burbuja` (49–56)
- `ordenamiento_seleccion` (59–68)

Detallado su funcionamiento en el apartado 5. Resultados Obtenidos

### Líneas 71–107: Interacción y presentación de resultados

```
71 # ----- Utilidades de interacción -----
72 def get_choice(prompt, min_val, max_val):
73     while True:
74         try:
75             choice = int(input(prompt))
76             if choice < min_val or choice > max_val:
77                 raise ValueError
78             return choice
79         except ValueError:
80             print(f"Debe seleccionar una opción entre {min_val} y {max_val}!")
81
82 # ----- Mostrar resultados ordenados automáticamente -----
83 def mostrar_resultados(df, campo_valor, campo_nombre='Nombre'):
84     if df.empty:
85         print("No se encontraron resultados que coincidan con el filtro.")
86         return
87
88     # Preparar lista de nombres y mapeo a filas
89     items = [(row[campo_nombre], row) for _, row in df.iterrows()]
90     nombres = [nombre for nombre, _ in items]
91     # Elegir algoritmo de ordenamiento según cantidad
92     if len(nombres) < 50:
93         nombres_ordenados = ordenamiento_burbuja(nombres)
94     else:
95         nombres_ordenados = ordenamiento_seleccion(nombres)
96     # Imprimir resultados en orden alfabético
97     print(f"Cantidad de letrados encontrados con el criterio seleccionado: {len(nombres_ordenados)}")
98     print("Lista con los datos de contacto:")
99     for idx, nombre in enumerate(nombres_ordenados, 1):
100         # Recuperar fila original
101         row = next(row for nom, row in items if nom == nombre)
102         valor = row[campo_valor] if pd.notna(row[campo_valor]) else 'No especificado'
103         telefono = row['Teléfono'] if pd.notna(row['Teléfono']) else 'No especificado'
104         direccion = row['Dirección'] if pd.notna(row['Dirección']) else 'No especificado'
105         etiqueta = campo_valor.replace('_', ' ')
106         print(f"{idx}) {nombre} - {etiqueta}: {valor}\n    Tel: {telefono} | Dir: {direccion}")
107
```

```
get_choice(prompt, min_val, max_val)
```

Es un bucle que pregunta al usuario un número, valida que esté dentro de un rango (por ejemplo, entre 1 y 4) y, si no, vuelve a preguntar. Así no se mete cualquier cosa.

```
mostrar_resultados(df, campo_valor, campo_nombre='Nombre')
```

Recibe el fragmento de la tabla con los abogados que ya cumplen un filtro, y dos nombres de columna: la que queremos usar para ordenar y la que muestra el “título” (por defecto, Nombre).

Transforma esa mini-tabla en una lista de nombres, decide qué algoritmo de ordenamiento usar según cuántos nombres haya (menos de 50 → burbuja; más → selección), y luego imprime cada nombre con sus datos de contacto (teléfono, dirección y el valor elegido: especialidad, años de experiencia, etc.).

Si no hay resultados, avisa “No se encontraron resultados...”.

## **Líneas 108–178: Lógica de filtrado, búsqueda automática y ejecución**

```
108 # ----- Lógica de filtrado -----
109 def filtrar_abogados(df):
110     print("En base a qué parámetros desea filtrar:")
111     print("1) Especialidad en el Derecho")
112     print("2) Años de experiencia")
113     print("3) Volumen de casos")
114     print("4) Formación académica (posgrados)")
115     filtro = get_choice("Seleccione una opción (1-4): ", 1, 4)
116
117     if filtro == 1:
118         for idx, esp in enumerate(especialidades, 1):
119             print(f"{idx}) {esp}")
120         seleccion = especialidades[get_choice(f"Ingrese número de 1 a {len(especialidades)}: ", 1, len(especialidades))]
121         df_filtrado = df[df['Especialidad'] == seleccion]
122         mostrar_resultados(df_filtrado, 'Especialidad')
123
124     elif filtro == 2:
125         min_exp = get_choice("Años mínimos de experiencia: ", 0, 100)
126         df_filtrado = df[df['Años de experiencia'] >= min_exp]
127         mostrar_resultados(df_filtrado, 'Años de experiencia')
128
129     elif filtro == 3:
130         min_casos = get_choice("Volumen mínimo de casos: ", 0, 10000)
131         if 'Volumen de casos' in df.columns:
132             df['Volumen de casos'] = pd.to_numeric(df['Volumen de casos'], errors='coerce').fillna(0)
133             df_filtrado = df[df['Volumen de casos'] >= min_casos]
134             df_filtrado = df_filtrado.sort_values(by='Volumen de casos', ascending=False)
135             mostrar_resultados(df_filtrado, 'Volumen de casos')
136         else:
137             print("La columna 'Volumen de casos' no existe en el archivo.")
138
139     else:
140         posibles_columnas = [col for col in df.columns if 'posgrado' in col.lower() or 'formación' in col.lower()]
141         if posibles_columnas:
142             columna = posibles_columnas[0]
143             print("¿Formación de posgrado en qué área del Derecho desea?")
144             for idx, esp in enumerate(especialidades, 1):
145                 print(f"{idx}) {esp}")
146             area = especialidades[get_choice(f"Seleccione área (1-{len(especialidades)}): ", 1, len(especialidades))]
147             df_filtrado = df[df[columna].str.contains(area, case=False, na=False)]
```

**filtrar\_abogados(df)**

Muestra un menú de 4 opciones (especialidad, años de experiencia, volumen de casos, posgrados).

Según la elección:

Pide más detalles (por ejemplo, qué especialidad exacta o cuántos años mínimos).

Crea una nueva mini-tabla (df\_filtrado) con solo los abogados que cumplen ese criterio.

Llama a mostrar\_resultados para imprimirlos.

**busqueda\_automatica(df)**

```
152 # ----- Búsqueda automática y despedida -----
153 def busqueda_automatica(df):
154     nombres = df['Nombre'].dropna().tolist()
155     palabra = input("\nIngrese nombre o parte del nombre para búsqueda automática: ")
156     # Intentar búsqueda binaria en lista ordenada
157     sorted_nombres = sorted(nombres)
158     idx = busqueda_binaria(sorted_nombres, palabra)
159     if idx != -1:
160         print(f"Encontrado en posición {idx+1}: {sorted_nombres[idx]}")
161     else:
162         idx = busqueda_lineal(nombres, palabra)
163         if idx != -1:
164             print(f"Encontrado en posición {idx+1}: {nombres[idx]}")
165         else:
166             print("No se encontró el abogado.")
167
168 # ----- Ejecución principal -----
169 if __name__ == "__main__":
170     df = cargar_datos()
171     if df.empty:
172         print("No se encontraron datos en el archivo.")
173         exit()
174
175     filtrar_abogados(df)
176     busqueda_automatica(df)
177     print("\nGracias por utilizar el sistema de búsqueda de abogados.\n¡Hasta luego!")
178
```

Después del filtrado, le da al usuario la opción de tipear parte del nombre de un abogado.

Intenta primero la búsqueda binaria sobre la lista ordenada de nombres y, si no lo encuentra, recurre a la búsqueda lineal.

Informa la posición en la lista o dice “No se encontró el abogado.”

### Bloque principal

if `__name__ == "__main__"`: le dice a Python: “Esto corre sólo si ejecutás este archivo directamente, no si lo importás desde otro programa.”

Dentro, primero carga los datos (`cargar_datos()`), verifica que haya algo, llama a `filtrar_abogados`, luego a `busqueda_automatica` y finalmente imprime un mensaje de despedida.

**ACLARACIÓN:** Cabe destacar que, para evitar confusiones o posibles dificultades con el usuario por desconocimientos técnicos respecto a las herramientas utilizadas y sus características, se evitó preguntar a los mismos que tipo de herramientas deseaba utilizar, dejando dicha tarea

directamente en manos del algoritmo, para que utilice las herramientas más adecuadas para cada circunstancia y cantidad de datos utilizados.

## 4. Metodología utilizada

### 4.1. Fuentes de información:

Para la revisión de material teórico relacionado al tema de este trabajo práctico que fuese de utilidad a la hora de aplicar los conceptos en el caso práctico se recurrió a distintos medios:

- Material teórico proporcionado por el equipo docente sobre temáticas de búsqueda y ordenamiento.

- Fue de vital importancia el PDF sobre búsqueda y ordenamiento que se encuentra dentro de la bibliografía de la unidad.

- Página web del colegio de abogados, a partir del cual se obtuvo la nómina de los letrados matriculados en el padrón enlistado en dicha página;

- Uso de inteligencia artificial (ChatGPT) para consultas técnicas e información específica sobre algunos temas.

### 4.2. Trabajo colaborativo

La investigación del caso se realizó en conjunto y, ante la imposibilidad física de juntarnos a hacer las pruebas, estas fueron realizadas en la computadora personal del hogar con participación activa del otro integrante a través de la plataforma Zoom, haciendo uso de las funciones de “Compartir Pantalla” para ir corroborando el avance en el proceso de testeo del código, como así también supervisar su modificación.

Una vez que cada función estuvo completa procedimos a escribir un código que las incluya a todas. Este código está adjunto con el nombre de: `tp_integrador_programacion1.py`

En el cual el usuario tiene la posibilidad de generar una lista de números aleatorios de tamaño a elección.

### Cronograma resumido de actividades

- 01 jun 2025: primera reunión por Zoom para planificar roles y descargar las fuentes de la cátedra.
- 02 jun 2025: Gastón entrega borrador del Marco Teórico; Diego empieza la ejecución de los modelos de prueba teóricos y a redactar el código para explicar el funcionamiento de cada método algorítmico estudiado.
- 03 jun 2025: Definición del tema práctico en el cual se aplicarían los conceptos teóricos, vía whatsapp.
- 06 junio 2025: Gaston busca la lista de letrados y su formación a nivel de posgrado; Diego redacta la sección de Metodología.
- 07 junio 2025: revisión final conjunta del documento en Word (edición colaborativa); generación de PDF para envío. Se termina de redactar el código sobre la lista de abogados y de corregir los errores (bugs) encontrados en su implementación práctica.
- 08 junio 2025: Se realiza la presentación power point y se procede a grabar y editar el video explicativo del tema escogido y su aplicación
- 09 jun 2025: Se sube todo al repositorio de GitHub y luego a la plataforma virtual en el espacio destinado a la entrega del trabajo.

## 5. Resultados obtenidos

Link al Repositorio de GitHub: [Click aquí](#)

En el proceso de redacción del código, experimentamos algunas dificultades en lo que respecta a las respuestas brindadas por el programa, ya que, en los primeros intentos, no lográbamos que el mismo se limite a entregar al usuario exclusivamente la información clave que dé respuesta a los parámetros de búsqueda escogidos, por lo que tuvimos que analizar cual era la mejor manera para optimizar dicho aspecto.

Asimismo, se utilizaron estructuras condicionales en bucle para que, en caso que el usuario introdujera cualquier información fuera de los parámetros u opciones otorgados, se repitiera la orden, aclarando cuales eran las opciones a escoger.

Más allá de las menciones efectuadas, estamos contentos con el resultado final obtenido, la



respuesta brindada por el código en concreto, como así también la necesidad y el potencial que esta herramienta ofrece para dar respuesta a una problemática social que hasta el día de hoy no había encontrado una respuesta idónea y práctica para vincular abogados con potenciales clientes y viceversa.

### 5.1. Tipos de algoritmos de búsqueda y ordenamiento utilizados

busqueda\_lineal (27–32)

busqueda\_binaria (35–46)

ordenamiento\_burbuja (49–56)

ordenamiento\_seleccion (59–68)

Contiene cuatro funciones clásicas, tomadas del material bibliográfico:

```
# ----- Algoritmos de búsqueda y ordenamiento -----
def busqueda_lineal(lista, elemento):
    # O(n)
    for i, v in enumerate(lista):
        if elemento.lower() in v.lower():
            return i
    return -1

def busqueda_binaria(lista, elemento):
    # O(log n), requiere lista ordenada
    izquierda, derecha = 0, len(lista) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista[medio].lower() == elemento.lower():
            return medio
        elif lista[medio].lower() < elemento.lower():
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1
```

#### 1. busqueda\_lineal (lista, elemento) (líneas 27–32)

- Recorre cada ítem de la lista buscando coincidencia de texto parcial (insensible a mayúsculas).

- Devuelve el índice de la primera coincidencia o -1 si no la encuentra.
- Complejidad  $O(n)$ : inspecciona todos los elementos en el peor caso.

## 2. `busqueda_binaria (lista, elemento)` (líneas 35–46)

- Divide el rango de búsqueda por la mitad en cada paso. Funciona solo si la lista está ordenada alfabéticamente.
- Retorna índice o -1. Complejidad  $O(\log n)$ .

```
def ordenamiento_burbuja(lista):  
    #  $O(n^2)$   
    arr = lista.copy()  
    for i in range(len(arr)):  
        for j in range(len(arr) - i - 1):  
            if arr[j].lower() > arr[j+1].lower():  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

```
def ordenamiento_seleccion(lista):  
    #  $O(n^2)$   
    arr = lista.copy()  
    for i in range(len(arr)):  
        min_idx = i  
        for j in range(i+1, len(arr)):  
            if arr[j].lower() < arr[min_idx].lower():  
                min_idx = j  
        arr[i], arr[min_idx] = arr[min_idx], arr[i]  
    return arr
```

## 3. `ordenamiento_burbuja (lista)` (líneas 49–56)

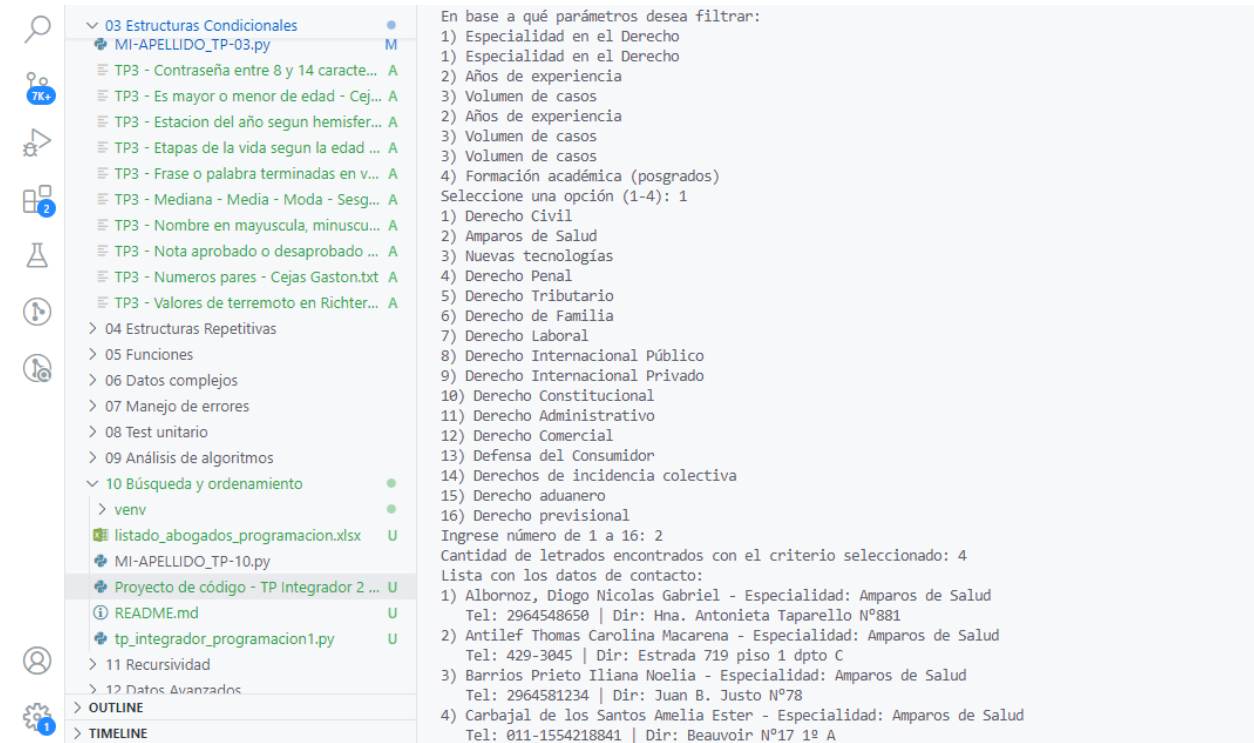
- Compara pares adyacentes y los intercambia si están fuera de orden, repitiendo pasadas hasta ordenar toda la lista.
- Complejidad  $O(n^2)$ .

#### 4. ordenamiento\_seleccion(lista) (líneas 59–68)

En cada iteración busca el elemento mínimo en la parte no ordenada y lo intercambia con la posición actual.

También  $O(n^2)$ .

Ejemplo del código en ejecución en la consola:



```

En base a qué parámetros desea filtrar:
1) Especialidad en el Derecho
2) Años de experiencia
3) Volumen de casos
4) Formación académica (posgrados)
Seleccione una opción (1-4): 1
1) Derecho Civil
2) Amparos de Salud
3) Nuevas tecnologías
4) Derecho Penal
5) Derecho Tributario
6) Derecho de Familia
7) Derecho Laboral
8) Derecho Internacional Público
9) Derecho Internacional Privado
10) Derecho Constitucional
11) Derecho Administrativo
12) Derecho Comercial
13) Defensa del Consumidor
14) Derechos de incidencia colectiva
15) Derecho aduanero
16) Derecho previsional
Ingrese número de 1 a 16: 2
Cantidad de letrados encontrados con el criterio seleccionado: 4
Lista con los datos de contacto:
1) Albornoz, Diogo Nicolas Gabriel - Especialidad: Amparos de Salud
Tel: 2964548650 | Dir: Hna. Antonieta Taparellero N°881
2) Antilef Thomas Carolina Macarena - Especialidad: Amparos de Salud
Tel: 429-3045 | Dir: Estrada 719 piso 1 dpto C
3) Barrios Prieto Iliana Noelia - Especialidad: Amparos de Salud
Tel: 2964581234 | Dir: Juan B. Justo N°78
4) Carbaljal de los Santos Amelia Ester - Especialidad: Amparos de Salud
Tel: 011-1554218841 | Dir: Beauvoir N°17 1º A
  
```

```

Ingrese nombre o parte del nombre para búsqueda automática: diogo
Encontrado en posición 5: Albornoz, Diogo Nicolas Gabriel
  
```

```

Gracias por utilizar el sistema de búsqueda de abogados.
¡Hasta luego!
(venv) PS C:\Users\Usuario\Desktop\Gaston\Programacion UTN\UTN-TUPaD-P1---Fork\10 Búsqueda y ordenamiento>
  
```

## 6. Conclusiones

Al finalizar este proyecto de aplicación de algoritmos de búsqueda y ordenamiento en Python para la gestión de un listado de abogados, el grupo de trabajo reflexiona sobre varios aprendizajes, utilidades, mejoras futuras y las dificultades encontradas:

### 1. Qué aprendimos al hacer el trabajo

Durante la implementación, profundizamos en el uso de estructuras de datos en Python (series y DataFrames de pandas), y comprendimos la importancia de elegir el algoritmo adecuado según el tamaño y características de la información. Pudimos comprobar empíricamente cómo la búsqueda binaria supera a la lineal en listas ordenadas y cómo, a pesar de su simplicidad, los métodos de burbuja y selección se vuelven prohibitivos cuando crece el volumen de datos.

## 2. Utilidad del tema para la programación y otros proyectos

- ✓ **Eficiencia y escalabilidad:** Los conceptos estudiados son esenciales en cualquier software que maneje grandes volúmenes de datos, desde aplicaciones web hasta sistemas de bases de datos.
- ✓ **Reutilización de código:** Las funciones desarrolladas pueden adaptarse fácilmente a otros contextos (por ejemplo, inventarios, catálogos de productos, listados de clientes).
- ✓ **Formación académica y profesional:** Dominar estos algoritmos es a menudo requisito en entrevistas técnicas y en la resolución de problemas reales donde el rendimiento es crítico.

## 3. Posibles mejoras y extensiones futuras

Implementar algoritmos avanzados: Incluir QuickSort o MergeSort para comparar su rendimiento con los métodos básicos.

Interfaz gráfica o web: Desarrollar un front-end que permita al usuario seleccionar criterios y visualizar resultados de forma más amigable.

Persistencia de datos: Conectar la aplicación a una base de datos SQL o NoSQL para gestionar cargas y actualizaciones masivas.

Análisis estadístico: Añadir gráficas de distribución de especialidades, experiencia y volumen de casos usando librerías como matplotlib o Plotly.

En conjunto, el proyecto no solo alcanzó su objetivo de permitir búsquedas y filtrados eficientes en un padrón de abogados, sino que también fortaleció nuestras competencias en

buenas prácticas de programación, manejo de dependencias y diseño de algoritmos escalables.

## 7. Bibliografía

-UTN Programación 1 (2025). Documentos compartidos en el Aula Virtual UTN – TUPaD – Material orientativo para la realización del trabajo integrador – Algoritmos de Búsqueda y Ordenamientos de Datos en Python.

- Padrón de abogados de la Provincia de Tierra del Fuego.

- Análisis de los Perfiles de LinkedIn de los profesionales.

## 8. Anexos

### a) Captura de la base de datos utilizada – Excel: listado\_abogados\_programación

1	Nombre	Especialidad	Posgrados	Años de experiencia	Volumen de casos	Teléfono	Dirección
2	Acevedo Alfredo Fabian	Derecho Internacional Público	Doctor en Nuevas tecnologías	14	57	2964641100	AV. 9 de julio N°959 planta baja
3	Acosta, Aidana Abigail	Derecho Tributario	Especialista en Derecho Constitucional; Magister en D	9	142	2964532870	Yewarsi N°345
4	Agüero Brancati, Leonardo Daniel Horacio	Derechos de incidencia colectiva	Magister en Derecho Comercial	7	33	1160559817	Pacheco N°742
5	Agüero Norma Gregoria	Derecho Penal	Doctor en Derecho Laboral	23	119	2964413906	Macizo 155 parcela 77 (tolhuin)
6	Albornoz, Diogo Nicolas Gabriel	Amparos de Salud	Magister en Nuevas tecnologías; Especialista en Dere	5	88	2964548650	Hna. Antonieta Taparello N°881
7	Alien Hector Guillermo	Derecho Comercial	Doctor en Derecho Constitucional; Magister en Derech	17	76	2964455057	Colon N° 953
8	Almiron Rina Noelia	Defensa del Consumidor	Especialista en Derecho de Familia	12	24	426-344	Bonocelli N°1269
9	Almonacid Navarro Cristina Isabel	Derecho Constitucional	Magister en Derecho Constitucional	27	103	15564738	Rio Chico N° 3629
10	Alvaredo Pablo Javier	Derecho Laboral	Doctor en Derecho Previsional; Especialista en Derech	3	46	2964-519896	Rosales N° 627
11	Alvares Andres	Derecho Internacional Privado	Magister en Nuevas tecnologías; Especialista en Dere	21	187	15616040	Pje. Trejo Noel N° 662
12	Alvarez Anita Isabel	Derecho Penal	Doctor en Defensa del Consumidor	19	92	2964552340	Alberdi 295 piso 1 of 3
13	Alvarez Laura Marcela	Nuevas tecnologías	Magister en Derecho Comercial; Especialista en Nueva	6	65	15457094	Piedra Buena N°2179
14	Alvarez Rivas, Vanesa Elizabeth	Derecho Civil	Doctor en Derecho Internacional Público	11	137	2964590396	Trejo Noel N°662
15	Alvarez Conde Guillermo	Derecho Administrativo	Especialista en Derecho Constitucional	8	54	2964611351	El cano N° 959
16	Alvarez Jose Luis	Derecho Penal	Magister en Derecho Penal	15	101	15402029	Estrada N° 400
17	Alvarez Mijail Emmanuel	Derecho Internacional Público	Doctor en Derecho Internacional Público	2	39	2964-411825	Alte. Brown 2278
18	Alvarez Manduani Marcelo Javier	Derecho Comercial	Magister en Derechos de incidencia colectiva	24	152	705 CPARG	Ameghino N° 654 local 2
19	Amadio Cristian Pablo	Derechos de incidencia colectiva	Doctor en Derecho Laboral	10	68	15530337	Estrada 495
20	Amarilla Orlando Ignacio	Derecho Penal	Especialista en Derecho Penal	5	43	434332	Lima 363 planta alta
21	Amaya Analía Belen	Derecho de Familia	Magister en Derecho de Familia	18	127	0260-154305565	Barrio YPF casa 724
22	Andino Carlos Ernesto	Derecho aduanero	Doctor en Nuevas tecnologías	3	57	421300-424806	Thorne N° 765
23	Andrade, Agustina	Derecho Internacional Privado	Doctor en Nuevas tecnologías	2	142	2964618615	Jose Romero N°2629
24	Andrade Hernandez Jose Octavio	Derecho Internacional Público	Doctor en Derecho de Familia	8	33	02964-478248	Almafuerte N° 434
25	Andrade Rosa Mandina	Derecho Constitucional	Magister en Derechos de incidencia colectiva; Especia	12	88	2964449518	Moyano N° 1381 Dpto. 1
26	Antilef Thomas Carolina Macarena	Amparos de Salud	Especialista en Derecho Procesal Penal	5	76	429-3045	Estrada 719 piso 1 dpto C
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							
41							
42							
43							
44							
45							
46							
47							
48							
49							
50							
51	Bierbrauer Walter	Defensa del Consumidor	Doctor en Derecho Constitucional; Especialista en Der	7	52	15-468-235	Perito Moreno N°1058
52	Bilbao Pablo Renán	Derecho Internacional Privado	Magister en Nuevas tecnologías; Doctor en Derecho Pe	12	143	426-699	Belgrano N°997
53	Bitsch Facundo	Derechos de incidencia colectiva	Especialista en Defensa del Consumidor	7	58	2964-15464555	Perito Moreno N°1435
54	Bitsch Martin	Derecho Procesal Penal	Doctor en Derecho Constitucional	19	121	433-679	Estrada N°740 Dpto 2 "A"
55	Blanco Nicolás Carlos	Derecho Penal	Magister en Derecho Comercial	5	34	1165257999	Roldan N°730
56	Bogado Jonatan Rene	Derecho Laboral	Especialista en Derecho Tributario o; Doctor en Derecho	23	177	—	Eva Perón N°624 Piso 1 – Dpto 6
57	Borra Rocio Natali	Derecho de Familia	Doctor en Amparos de Salud	16	92	2964-602065	Julio Popper N°544
58	Bravo Mariela Aida	Defensa del Consumidor	Magister en Nuevas tecnologías	9	47	424088	Ushuaia N°786
59	Breitenbacher Guillermo Juan	Derecho Constitucional	Especialista en Derechos de incidencia colectiva	28	163	504367	Visis N°2841
60	Bres Alexia Romina	Derecho Civil	Doctor en Derecho Internacional Público	4	21	443750	Provincias Unidas N°884
61	Brites Cristian Mariano	Derecho Comercial	Especialista en Derecho Tributario; Magister en Derech	14	136	0294-154706551	Metet N°392, Tolhuin
62	Bucci Matias Ezequiel	Derecho Internacional Público	Doctor en Derecho Penal	11	87	—	—
63	Burgos Karina Soledad Vanesa	Derecho Laboral	Magister en Defensa del Consumidor; Especialista en	7	53	426-971	Darwin N° 481
64	Bustos Lopez Jose Maria	Derecho Penal	Especialista en Derecho Procesal Penal	15	142	—	Peru N° 85
65	Bustos Varela Florencia Romina	Nuevas tecnologías	Doctor en Nuevas tecnologías; Magister en Derecho Co	4	29	2964465566	Bahia San Julian 470
66	Buzzierio Federico Matias	Defensa del Consumidor	Magister en Derecho Laboral	19	113	2964402029	Estrada N° 400
67	Cafiete Sonia Mabel	Derecho Constitucional	Magister en Derecho Laboral	17	78	2964539396	11 de Julio 1284
68	Campos Cesia Silvana	Derecho Tributario	Especialista en Derecho Comercial	5	150	2964-560510	Irigoyen N°1037
69	Campos Mauro	Derechos de incidencia colectiva	Especialista en Defensa del Consumidor	12	60	2964533345	Don Bosco N°1037
70	Campos Nuñez Daiana Celeste	Derecho Laboral	Doctor en Derecho Penal	3	45	2964543816	Jose Romero N°2895
71	Capotorto Valeria Andrea	Derecho Comercial	Magister en Derecho Administrativo	20	200	423-847	O'Higgins N° 194
72	Carbajal de los Santos Amelia Ester	Amparos de Salud	Doctor en Nuevas tecnologías	8	55	011-1554218841	Beauvoir N°17 1º A
73	Carballo Gonzalo	Derecho Internacional Privado	Especialista en Derecho Internacional Público	10	95	15-579-359	Federico Echelaine N°185
74	Cardenas Elio Ruben	Derecho Previsional	Magister en Derecho Constitucional	25	110	434332	Cirilo Tomas N°782 B° AGP
75	Cardenia Myrian Raquel	Derecho Civil	Especialista en Derecho Penal	14	85	15-511-757	Salidas N°919
76	Cardozo Claudia Marcela	Defensa del Consumidor	Doctor en Derecho Tributario	6	70	2964590825	Gdor. Trejo Noel N°795

## b) Capturas del Código funcionando

```
eda y ordenamiento/Proyecto de código - IP Integrador 2 - Lista de abogados 4.py"
En base a qué parámetros desea filtrar:
1) Especialidad en el Derecho
2) Años de experiencia
3) Volumen de casos
4) Formación académica (posgrados)
Seleccione una opción (1-4): 5
Debe seleccionar una opción entre 1 y 4!
Seleccione una opción (1-4): 1
1) Derecho Civil
2) Amparos de Salud
3) Nuevas tecnologías
4) Derecho Penal
5) Derecho Tributario
6) Derecho de Familia
7) Derecho Laboral
8) Derecho Internacional Público
9) Derecho Internacional Privado
10) Derecho Constitucional
11) Derecho Administrativo
12) Derecho Comercial
13) Defensa del Consumidor
14) Derechos de incidencia colectiva
15) Derecho aduanero
16) Derecho previsional
Ingrese número de 1 a 16: 2
Cantidad de letrados encontrados con el criterio seleccionado: 4
Lista con los datos de contacto:
1) Albornoz, Diogo Nicolas Gabriel - Especialidad: Amparos de Salud
  Tel: 2964548650 | Dir: Hna. Antonieta Taparello N°881
2) Antilef Thomas Carolina Macarena - Especialidad: Amparos de Salud
  Tel: 429-3045 | Dir: Estrada 719 piso 1 dpto C
3) Barrios Prieto Iliana Noelia - Especialidad: Amparos de Salud
  Tel: 2964581234 | Dir: Juan B. Justo N°78
4) Carbajal de los Santos Amelia Ester - Especialidad: Amparos de Salud
  Tel: 011-1554218841 | Dir: Beauvoir N°17 1º A

Ingrese nombre o parte del nombre para búsqueda automática: 
```

6 unknown 10 hrs

## c) Cuadro comparativo entre los trabajos a medida que íbamos avanzando en su desarrollo

Entre los dos scripts subidos, el “Lista de abogados 4.py” y el “Lista de abogados.py”, el más completo y mejor alineado con los criterios de la cátedra (según el material “Búsqueda y

Ordenamiento en Programación 1”) es Lista de abogados 4.py. A continuación, un desglose de por qué:

Criterio	Lista de abogados 4.py	Lista de abogados.py
Cobertura de algoritmos de búsqueda	Incluye búsqueda lineal y binaria, y elige automáticamente entre ambos (binaria sobre lista ordenada, fallback lineal).	Solo hace búsqueda binaria seguida de lineal si falla, pero deja al usuario elegir en versiones previas y no integra orden de llamada automático.
Cobertura de algoritmos de ordenamiento	Implementa bubble sort y selection sort, eligiendo el más adecuado según tamaño de lista, y los usa internamente para presentar resultados ordenados.	No incluye los algoritmos clásicos de ordenamiento (solo ordena por Pandas o solicitaba manualmente al usuario).
Modularidad y claridad	Está claramente seccionado en bloques: carga, algoritmos, utilidades, despliegue, filtrado y búsqueda. Las funciones tienen comentarios de complejidad y propósito.	Modular, pero con menos separación de responsabilidades y sin comentarios de complejidad.
Adherencia al material bibliográfico	Usa directamente los algoritmos enseñados en el pdf (búsqueda lineal, binaria, bubble, selección) y automatiza su elección según contexto.	Solo implementa búsqueda lineal y binaria; no aprovecha los ordenamientos vistos.
Experiencia de usuario	Oculto detalles de “qué algoritmo” y ofrece un flujo más limpio: pide filtro, muestra resultados ordenados, pide búsqueda y despide.	Presenta opciones adicionales y detalle de elección de algoritmo que, aunque útil académicamente, puede confundir al usuario real.
Manejo de datos con Pandas	Usa Pandas para cargar y filtrar, pero delega el ordenamiento a los algoritmos clásicos, demostrando integración de ambos mundos.	Depende casi exclusivamente de Pandas (sort_values) para ordenar, lo cual es correcto pero no demuestra los algoritmos clásicos.
Robustez y control de errores	Controla la ausencia de archivo y de columnas de posgrado o volumen de casos, con mensajes claros.	Similar manejo de errores en carga y filtrado, pero sin comentarios explicativos ni modularidad extra.

**d) Enlaces a las partes del trabajo:**

- 1) [Repositorio en Github](#)
- 2) [Documento d el Trabajo Integrador Documentado](#)
- 3) [Enlace al video de Youtube](#) o también [Opcion 2 \(Haz click aquí\)](#)
- 4) [Base de datos utilizada: Padron de Abogados](#)
- 5) [Presentacion Power Point utilizada](#)
- 6) [Programa - Teoría - Diego](#)
- 7) [Programa - Practica - Padron de abogados](#)
- 8) [Archivo README](#)