Trabajo Integrador 1 – Arquitectura y Sistemas Operativos

- **Tema**: Algoritmos de Búsqueda y Ordenamiento en Python
- **Alumnos**: Matías Roda - matute.22@live.com.ar
  Matías Rodríguez - xmatiasrcx@gmail.com
- **Materia**: Programación I
- **Comisión**: Nro. 9
- **Profesor/a**: AUS Bruselario, Sebastián
- **Tutor/a**: Gubiotti, Flor

- **Índice**

1. Introducción

En la programación, los algoritmos de búsqueda y ordenamiento son fundamentales para la gestión eficiente de grandes volúmenes de datos. Comprender sus implementaciones y casos de uso es clave para desarrollar software rápido y escalable. En este trabajo, se analiza un sistema de gestión de legajos de estudiantes, donde, a partir de una lista de números de legajo, se comparan tres algoritmos de búsqueda —lineal, binaria iterativa y binaria recursiva— en diferentes escenarios de tamaño y tipos de casos (mejor, peor, aleatorios y fallido). Se presta especial atención a la búsqueda binaria por su relevancia en estructuras de datos previamente ordenadas y su impacto en el rendimiento en aplicaciones reales.

---

2. Marco Teórico

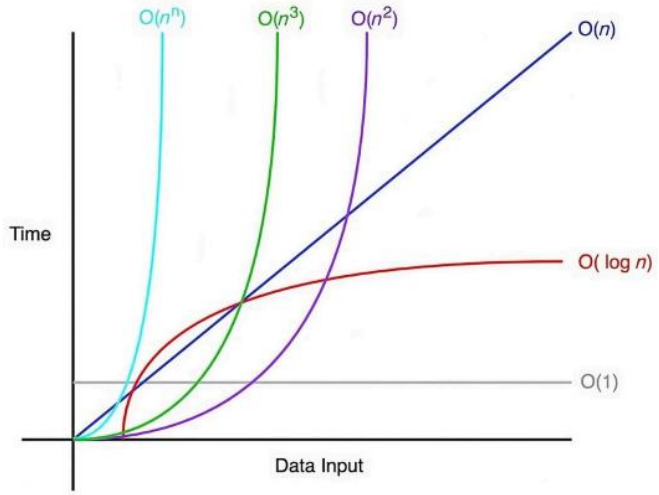
Este apartado contiene la fundamentación conceptual de los algoritmos de búsqueda y ordenamiento, incluyendo definiciones formales, clasificación de métodos, estructuras de datos involucradas y ejemplos de sintaxis en Python:

- **Búsqueda:** operación que localiza un elemento objetivo en una colección de datos
  - **Lineal:** recorre secuencialmente todos los elementos hasta encontrar el deseado.
  - **Binaria:** eficiente en listas ordenadas, divide el espacio de búsqueda en mitades
    sucesivas.
- **Ordenamiento:** permite reorganizar una colección de elementos de acuerdo a un criterio específico (mayor a menor, alfabéticamente, etc.)
  - **Bubble Sort (Método burbuja):** compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto.
  - **Quick Sort (Ordenamiento rápido):** selecciona un "pivote" y organiza los elementos menores a un lado y los mayores al otro.
- **Complejidad algoritmos:** La complejidad medida con O(n) es una forma de medir la eficiencia de un algoritmo. Representa el tiempo que tarda un algoritmo en ejecutarse en función del tamaño de la entrada principal.

La notación O(n) se utiliza para describir el peor caso de complejidad de tiempo

de un algoritmo. Esto significa que el algoritmo nunca tardará más de O(n) tiempo en ejecutarse para cualquier entrada de tamaño n. La complejidad medida con O(n) es una herramienta útil para comparar diferentes algoritmos y elegir el más eficiente para una tarea determinada.

Los algoritmos de búsqueda lineal tienen un tiempo de ejecución de O(n), lo que significa que el tiempo de búsqueda es directamente proporcional al tamaño de la lista y los algoritmos de búsqueda binaria tienen un tiempo de ejecución de O(log n), lo que significa que el tiempo de búsqueda aumenta logarítmicamente con el tamaño de la lista.



3. Caso Práctico

En esta caso práctico vamos a realizar un código para unas listas de legajos de la Universidad Tecnológica en el que dada una listas de legajos(mejor, peor, aleatorios y fallido) nos informe la posición del legajo en caso de que lo encuentre y el tiempo que se demoró en buscar. Se puede elegir 3 listas:

1. Alumnos de la materia programación 1 (200 legajos)
2. Alumnos Tecnicatura Universitaria en programación (20000 legajos)
3. Alumnos de la UTN (200000 legajos)

Cada lista de legajos se genera aleatoriamente (sin duplicados) y, antes de aplicar búsqueda estas se ordenan con Método burbuja (Bubble Sort) para la primer lista y Ordenamiento rápido (Quicksort) para las otras dos.

Con este programa podemos comparar tiempos métodos de búsqueda con distintos tamaños de lista para ver cuál es el más eficiente.

Código y explicación

- Las librerías utilizadas son random para generar las listas sin repetir con la función “random.sample()” y la librería time para cronometrar con la función “time.perf_counter()”. Aquí usamos dos tipos de búsqueda bubble sort para la lista mas pequeña y para las listas más grandes Quicksort.

```python
# Librerias
import random
import time

# Algoritmo de Ordenamientos
# 1. Ordenamiento burbuja (Bubble Sort)
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

# 2. Ordenamiento Rápido (Quicksort). Para Listas grandes
def quicksort(lista):
    if len(lista) <= 1:
        return lista
    else:
        pivot = lista[0]
        less = [x for x in lista[1:] if x <= pivot]
        greater = [x for x in lista[1:] if x > pivot]
        return quicksort(less) + [pivot] + quicksort(greater)
```

- Los métodos de búsqueda utilizados (lineal, binaria interactiva y recursiva)

```python
# Algoritmos de Búsqueda
# 1. Búsqueda Lineal
def busqueda_lineal(lista, objetivo):
    for i in range(len(lista)):
        if lista[i] == objetivo:
            return i
    return -1

# 2.1 Búsqueda Binaria
def busqueda_binaria(lista, objetivo):
    izquierda, derecha = 0, len(lista) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista[medio] == objetivo:
            return medio
        elif lista[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1

# 2.2 Búsqueda Binaria Recursiva
def busqueda_binaria_recursiva(lista, objetivo, izquierda, derecha):
    if izquierda > derecha:
        return -1
    medio = (izquierda + derecha) // 2
    if lista[medio] == objetivo:
        return medio
    elif lista[medio] < objetivo:
        return busqueda_binaria_recursiva(lista, objetivo, medio + 1, derecha)
    else:
        return busqueda_binaria_recursiva(lista, objetivo, izquierda, medio - 1)

```

- Menú de opciones.

```
60  # Inicio del programa
61  print("- Alumnos de Tecnicatura Universitaria en Programación")
62  op = input("Ingrese su opción (1: Legajos Alumnos Programacion I, 2: Legajos Alumnos Tecnicatura, 3: Legajos Alumnos Universidad): ")
63
64  if op == "1": ···
103
104 elif op == "2": ···
146
147 elif op == "3": ···
190
191 else:
192     print("Opción no válida")
```

- Legajos de alumnos programación 1 (200 legajos).

```
64  if op == "1":
65      # Generación de legajos únicos y eleccion de un objetivo
66      legajos = random.sample(range(100, 10000), 200)         # Genera 200 legajos únicos entre 100 y 9999
67      bubble_sort(legajos)                                    # Aplica el ordenamiento burbuja
68
69      # Eleccion de legajos a buscar
70      buscar_legajo = [
71          random.choice(legajos),         # Elige un elemento al azar
72          legajos[-1],                    # Elige un elemento al final de la lista
73          legajos[0],                     # Elige un elemento al inicio de la lista
74          legajos[len(legajos) // 2],     # Elige un elemento en el medio de la lista
75          max(legajos) + 1                 # Agrega un legajo que no está en la lista para probar la búsqueda fallida
76
77      for i in range(len(buscar_legajo)):
78          print(f"* Legajos a buscar: {buscar_legajo[i]}")               # Imprime los legajos generados
79          # Busqueda Lineal
80          t0 = time.perf_counter()                                # Inicia el cronómetro
81          resultado_lineal = busqueda_lineal(legajos, buscar_legajo[i])
82          t1 = time.perf_counter()                                # Detiene el cronómetro
83          tiempo_lineal = t1 - t0
84          print(f"- Búsqueda Lineal → índice: {resultado_lineal}, tiempo: {tiempo_lineal:.8f} s")
85
86          # Búsqueda Binaria
87          t0 = time.perf_counter()                                # Inicia el cronómetro
88          resultado_binaria = busqueda_binaria(legajos, buscar_legajo[i])
89          t1 = time.perf_counter()                                # Detiene el cronómetro
90          tiempo_binaria = t1 - t0
91          print(f"- Búsqueda Binaria → índice: {resultado_binaria}, tiempo: {tiempo_binaria:.8f} s")
92
93          # Búsqueda Binaria Recursiva
94          t0 = time.perf_counter()                                # Inicia el cronómetro
95          resultado_binaria_recursiva = busqueda_binaria_recursiva(legajos, buscar_legajo[i], 0, len(legajos) - 1)
96          t1 = time.perf_counter()                                # Detiene el cronómetro
97          tiempo_binaria_recursiva = t1 - t0
98          print(f"- Búsqueda Binaria Recursiva → índice: {resultado_binaria_recursiva}, tiempo: {tiempo_binaria_recursiva:.8f} s \n")
```

- Legajos de alumnos de Tecnicatura Universitaria en programación (20000 legajos).

```python
elif op == "2":
    # Generación de legajos únicos y eleccion de un objetivo
    legajos = random.sample(range(100, 100000), 20000)       # Genera 20000 legajos únicos entre 100 y 99999
    legajos= quicksort(legajos)                              # Aplica el ordenamiento rápido

    # Eleccion de legajos a buscar
    buscar_legajo = [
        random.choice(legajos),         # Elige un elemento al azar
        legajos[-1],                    # Elige un elemento al final de la lista
        legajos[0],                     # Elige un elemento al inicio de la lista
        legajos[len(legajos) // 2],     # Elige un elemento en el medio de la lista
        max(legajos) + 1]                # Agrega un legajo que no está en la lista para probar la búsqueda fallida

    for i in range(len(buscar_legajo)):  (variable) buscar_legajo: list
        print(f"* Legajos a buscar: {buscar_legajo[i]}")              # Imprime los legajos generados
        # Busqueda Lineal
        t0 = time.perf_counter()                                    # Inicia el cronómetro
        resultado_lineal = busqueda_lineal(legajos, buscar_legajo[i])
        t1 = time.perf_counter()                                    # Detiene el cronómetro
        tiempo_lineal = t1 - t0
        print(f"- Búsqueda Lineal → índice: {resultado_lineal}, tiempo: {tiempo_lineal:.8f} s")

        # Búsqueda Binaria
        t0 = time.perf_counter()                                   # Inicia el cronómetro
        resultado_binaria = busqueda_binaria(legajos, buscar_legajo[i])
        t1 = time.perf_counter()                                   # Detiene el cronómetro
        tiempo_binaria = t1 - t0
        print(f"- Búsqueda Binaria → índice: {resultado_binaria}, tiempo: {tiempo_binaria:.8f} s")

        # Búsqueda Binaria Recursiva
        t0 = time.perf_counter()                                   # Inicia el cronómetro
        resultado_binaria_recursiva = busqueda_binaria_recursiva(legajos, buscar_legajo[i], 0, len(legajos) - 1)
        t1 = time.perf_counter()                                   # Detiene el cronómetro
        tiempo_binaria_recursiva = t1 - t0
        print(f"- Búsqueda Binaria Recursiva → índice: {resultado_binaria_recursiva}, tiempo: {tiempo_binaria_recursiva:.8f} s \n")

```

- Legajos de alumnos de la UTN (200000 legajos).

```python
elif op == "3":
    # Generación de legajos únicos 200000 legajos únicos entre 100 y 499999
    legajos = random.sample(range(100, 500000), 200000)
    legajos= quicksort(legajos)

    # Eleccion de un legajo
    buscar_legajo = [
        random.choice(legajos),         # Elige un elemento al azar
        legajos[-1],                    # Elige un elemento al final de la lista
        legajos[0],                     # Elige un elemento al inicio de la lista
        legajos[len(legajos) // 2],     # Elige un elemento en el medio de la lista
        max(legajos) + 1]                # Agrega un legajo que no está en la lista para probar la búsqueda fallida

    for i in range(len(buscar_legajo)):
        print(f"* Legajos a buscar: {buscar_legajo[i]}")               # Imprime los legajos generados
        # Busqueda Lineal
        t0 = time.perf_counter()                                    # Inicia el cronómetro
        resultado_lineal = busqueda_lineal(legajos, buscar_legajo[i])
        t1 = time.perf_counter()                                    # Detiene el cronómetro
        tiempo_lineal = t1 - t0
        print(f"- Búsqueda Lineal → índice: {resultado_lineal}, tiempo: {tiempo_lineal:.8f} s")
        # Búsqueda Binaria
        t0 = time.perf_counter()                                   # Inicia el cronómetro
        resultado_binaria = busqueda_binaria(legajos, buscar_legajo[i])
        t1 = time.perf_counter()                                   # Detiene el cronómetro
        tiempo_binaria = t1 - t0
        print(f"- Búsqueda Binaria → índice: {resultado_binaria}, tiempo: {tiempo_binaria:.8f} s")
        # Búsqueda Binaria Recursiva
        t0 = time.perf_counter()                                   # Inicia el cronómetro
        resultado_binaria_recursiva = busqueda_binaria_recursiva(legajos, buscar_legajo[i], 0, len(legajos) - 1)
        t1 = time.perf_counter()                                   # Detiene el cronómetro
        tiempo_binaria_recursiva = t1 - t0
        print(f"- Búsqueda Binaria Recursiva → índice: {resultado_binaria_recursiva}, tiempo: {tiempo_binaria_recursiva:.8f} s \n")
```

Ejecución y resultados

Ejecución de las tres opciones de lista, con los diferentes casos el primero un numero aleatorio, ultimo elemento de la lista, primer elemento, elemento intermedio y uno que no esté en la lista.

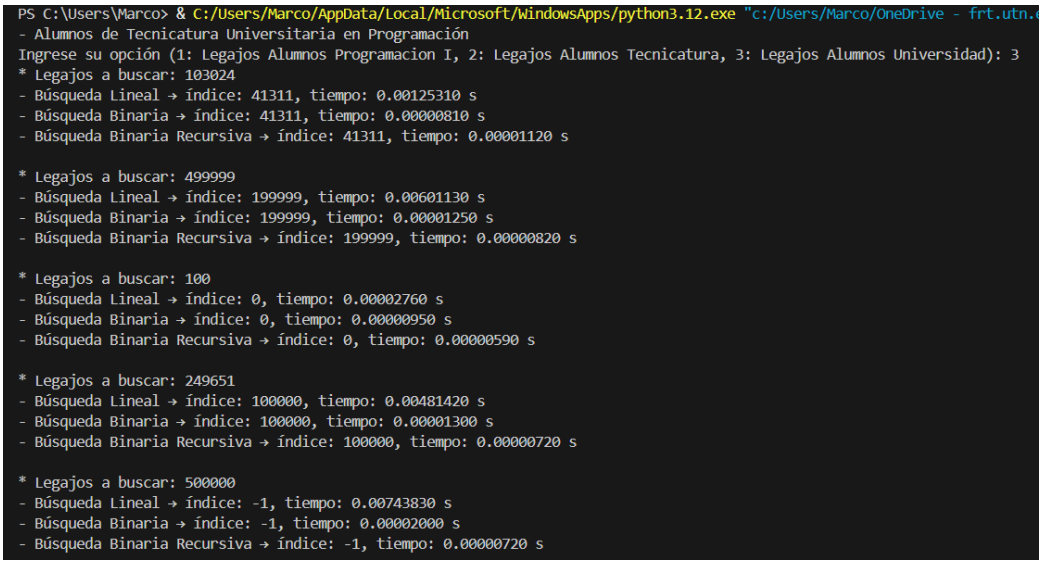
- Lista de 200 legajos.

```
PS C:\Users\Marco> & C:/Users/Marco/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Marco/OneDrive - frt.utn.edu.a
- Alumnos de Tecnicatura Universitaria en Programación
Ingrese su opción (1: Legajos Alumnos Programacion I, 2: Legajos Alumnos Tecnicatura, 3: Legajos Alumnos Universidad): 1
* Legajos a buscar: 1629
- Búsqueda Lineal → índice: 42, tiempo: 0.00000870 s
- Búsqueda Binaria → índice: 42, tiempo: 0.00000610 s
- Búsqueda Binaria Recursiva → índice: 42, tiempo: 0.00000670 s

* Legajos a buscar: 9898
- Búsqueda Lineal → índice: 199, tiempo: 0.00001570 s
- Búsqueda Binaria → índice: 199, tiempo: 0.00000660 s
- Búsqueda Binaria Recursiva → índice: 199, tiempo: 0.00000480 s

* Legajos a buscar: 104
- Búsqueda Lineal → índice: 0, tiempo: 0.00000310 s
- Búsqueda Binaria → índice: 0, tiempo: 0.00000450 s
- Búsqueda Binaria Recursiva → índice: 0, tiempo: 0.00000390 s

* Legajos a buscar: 4765
- Búsqueda Lineal → índice: 100, tiempo: 0.00000630 s
- Búsqueda Binaria → índice: 100, tiempo: 0.00000610 s
- Búsqueda Binaria Recursiva → índice: 100, tiempo: 0.00000720 s

* Legajos a buscar: 9899
- Búsqueda Lineal → índice: -1, tiempo: 0.00001480 s
- Búsqueda Binaria → índice: -1, tiempo: 0.00000650 s
- Búsqueda Binaria Recursiva → índice: -1, tiempo: 0.00000710 s
```

- Lista de 20000 legajos.

```
PS C:\Users\Marco> & C:/Users/Marco/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Marco/OneDrive - frt.utn.
- Alumnos de Tecnicatura Universitaria en Programación
Ingrese su opción (1: Legajos Alumnos Programacion I, 2: Legajos Alumnos Tecnicatura, 3: Legajos Alumnos Universidad): 2
* Legajos a buscar: 12411
- Búsqueda Lineal → índice: 2442, tiempo: 0.00010190 s
- Búsqueda Binaria → índice: 2442, tiempo: 0.00000710 s
- Búsqueda Binaria Recursiva → índice: 2442, tiempo: 0.00000740 s

* Legajos a buscar: 99992
- Búsqueda Lineal → índice: 19999, tiempo: 0.00128990 s
- Búsqueda Binaria → índice: 19999, tiempo: 0.00000660 s
- Búsqueda Binaria Recursiva → índice: 19999, tiempo: 0.00000920 s

* Legajos a buscar: 101
- Búsqueda Lineal → índice: 0, tiempo: 0.00000310 s
- Búsqueda Binaria → índice: 0, tiempo: 0.00000600 s
- Búsqueda Binaria Recursiva → índice: 0, tiempo: 0.00000990 s

* Legajos a buscar: 50214
- Búsqueda Lineal → índice: 10000, tiempo: 0.00041810 s
- Búsqueda Binaria → índice: 10000, tiempo: 0.00000460 s
- Búsqueda Binaria Recursiva → índice: 10000, tiempo: 0.00000600 s

* Legajos a buscar: 99993
- Búsqueda Lineal → índice: -1, tiempo: 0.00120200 s
- Búsqueda Binaria → índice: -1, tiempo: 0.00000780 s
- Búsqueda Binaria Recursiva → índice: -1, tiempo: 0.00000830 s
```

- Lista de 200000 legajos



Resultados

Como podemos observar la búsqueda lineal crece en proporción al tamaño de la lista y al caso, siendo el mejor caso (elemento al inicio) rápido, pero en el peor caso (último o inexistente) muy lento en listas grandes. En cambio la búsqueda binaria (interactiva y recursiva) se mantienen constante al aumentar la cantidad de elementos.

La diferencia de tiempo entre la búsqueda interactiva y recursiva es apenas perceptible.

En conclusión para listas pequeños la diferencia es apenas notable; la lineal es aceptable. Para listas grandes la búsqueda binaria es claramente superior, ofreciendo respuestas rápidas.

4. Metodología Utilizada

Para el desarrollo del presente trabajo se siguieron una serie de pasos organizados en distintas etapas, con el objetivo de aplicar los conceptos teóricos de búsqueda y ordenamiento en un entorno práctico.

- **Investigación previa**

  El trabajo de investigación comenzó con el análisis del material de la unidad de búsqueda y ordenamiento disponible en el campus virtual, incluyendo tanto los videos explicativos como la teoría proporcionada por la cátedra.
  Para profundizar los conceptos y mejorar la implementación, se complementó con la consulta de artículos y videotutoriales externos.
  De esta manera, la investigación previa combinó recursos teóricos de la materia con contenido adicional encontrado en plataformas digitales, permitiendo comprender mejor las buenas prácticas y aplicar técnicas adecuadas al tema seleccionado.

- **Etapas de diseño y prueba del código**

  Se desarrolló un programa que genera automáticamente tres listas de legajos de diferente tamaño para evaluar la escalabilidad de los algoritmos. Cada lista se crea sin duplicados y, mediante un menú interactivo, el usuario puede elegir el tamaño deseado y ejecutar cinco escenarios de búsqueda (aleatorio, inicio, medio, final e inexistente). Con “time.perf_counter()” se registra el tiempo de las búsquedas lineal, binaria iterativa y recursiva, lo que permite comparar de forma precisa su rendimiento en cada caso.

- **Herramientas y recursos utilizados**
  - Python 3.11.9 como lenguaje de programación
  - Visual Studio Code como editor de código
  - Capturas de pantalla y el grabador de pantalla del sistema para la documentación.

- **Trabajo colaborativo**

  El trabajo colaborativo fue desarrollado dividiendo el trabajo en partes, en donde una se encargó de documentar y de generar el documento técnico y la otra parte en el desarrollo y generación del programa. La comunicación fue constante mediante reuniones virtuales (Discord) y mensajes (WhatsApp), también el uso de una carpeta compartida.

5. Resultados Obtenidos

Con este trabajo se logró crear tres listas de datos (200, 20000 y 200000 legajos) aleatorios sin repetirse, su ordenamiento con bubble sort para la lista de 200 legajos y las otras dos se usó el método Quicksort.

Para cada lista tres métodos de búsqueda lineal, binaria iterativa y recursiva y con cinco escenarios de búsqueda: **Aleatorio, Primer elemento, Elemento del medio, Elemento Final y elemento fuera del rango.**

Cada método tiene un cronometro para comparar los tiempos con los diferentes tipos de búsqueda.

Con esto se comprobó que la búsqueda lineal no es la más óptima para listas grandes ya que el tiempo crece en proporción al tamaño de la lista y el peor caso (último elemento o fuera de rango) su demora incrementa muchísimo. En cambio los métodos binarios mantienen el rango de tiempo de forma logarítmica sin incrementar mucho, ni en el peor caso.

Algunos errores o correcciones que hicimos es el método de ordenamiento inicialmente se usó bubble Sort para las tres lista y resulto demasiado lento.

También se usó la función time.perf_counter() para mejorar la precisión en rangos de microsegundos.

El proyecto completo, con documentación y capturas de pantalla, está disponible en GitHub: https://github.com/MatiasR85/TP-INTEGRADOR1

6. Conclusiones

Al desarrollar este trabajo se confirma que la búsqueda binaria es la opción óptima para grandes volúmenes de datos, y la experiencia práctica refuerza los conceptos teóricos de complejidad algorítmica aprendidos en la materia. También aprendimos a implementar y comparar distintos algoritmos de búsqueda y ordenamiento en Python, entendiendo de manera práctica por qué la búsqueda lineal crece linealmente mientras que la binaria se mantiene muy eficiente incluso al multiplicar por mil el tamaño de la lista. El tema resulta especialmente útil en cualquier sistema que requiera localizar datos rápidamente por ejemplo, en bases de datos donde la elección del algoritmo y la preparación previa de los datos (ordenamiento) marcan la diferencia en rendimiento. Como avance de este trabajo podríamos comparar los

distintos tipos de ordenamientos y también otros métodos de búsqueda como tablas Hash.

7. Bibliografía

- Unidad Búsqueda y Ordenamiento. Programación 1. Aula Virtual.
- BÚSQUEDA BINARIA con Python 3 - Clase #56. YouTube. https://www.youtube.com/watch?v=6-e_kVZkECI
- https://docs.python.org/es/3.10/library/time.html. Librería Time. Manual de Python.