**Trabajo Integrador:** Búsqueda y ordenamiento

**Alumno:**

Gonzalo Emanuel Nuñez [- gonza\_emmanuel@hormail.com.ar](mailto:-%20gonza_emmanuel@hormail.com.ar)

**Índice**

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía

8. Anexos

**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, incluyendo una explicación especial sobre la búsqueda binaria, por su relevancia en estructuras de datos ordenadas

**Marco Teórico**

**Bubble Sort (Ordenamiento Burbuja)**

El **algoritmo de ordenamiento burbuja** compara elementos adyacentes en una lista y los intercambia si están en el orden incorrecto. Repite este proceso varias veces hasta que la lista está ordenada. Se llama "burbuja" porque los valores más grandes van “flotando” hacia el final de la lista con cada pasada.

Existen dos versiones:

* **Clásica**: hace todas las pasadas, aunque ya esté ordenada.
* **Optimizada (corta)**: se detiene si en una pasada no hay intercambios, mejorando el rendimiento.

**Búsqueda Binaria**

La **búsqueda binaria** es un algoritmo eficiente para encontrar un valor dentro de una **lista ordenada**. Divide la lista a la mitad y compara el valor buscado con el elemento central. Si no coincide, decide si buscar en la mitad izquierda o derecha, repitiendo el proceso hasta encontrar el valor o agotar la lista.

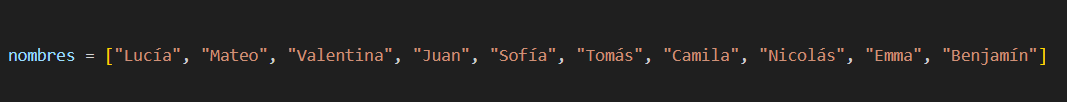
Tiene una **complejidad O (log n)**, lo que la hace mucho más rápida que la búsqueda secuencial para listas grandes, pero **requiere que la lista esté previamente ordenada**.

**Caso Practico**

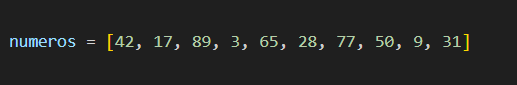
Tenemos 2 listas en una donde tenemos nombres y otra de números aleatorios, en donde aplicaremos en Python el método de ordenamiento Bubble Sort (Optima) y búsqueda secuencial.

Listas desordenadas =

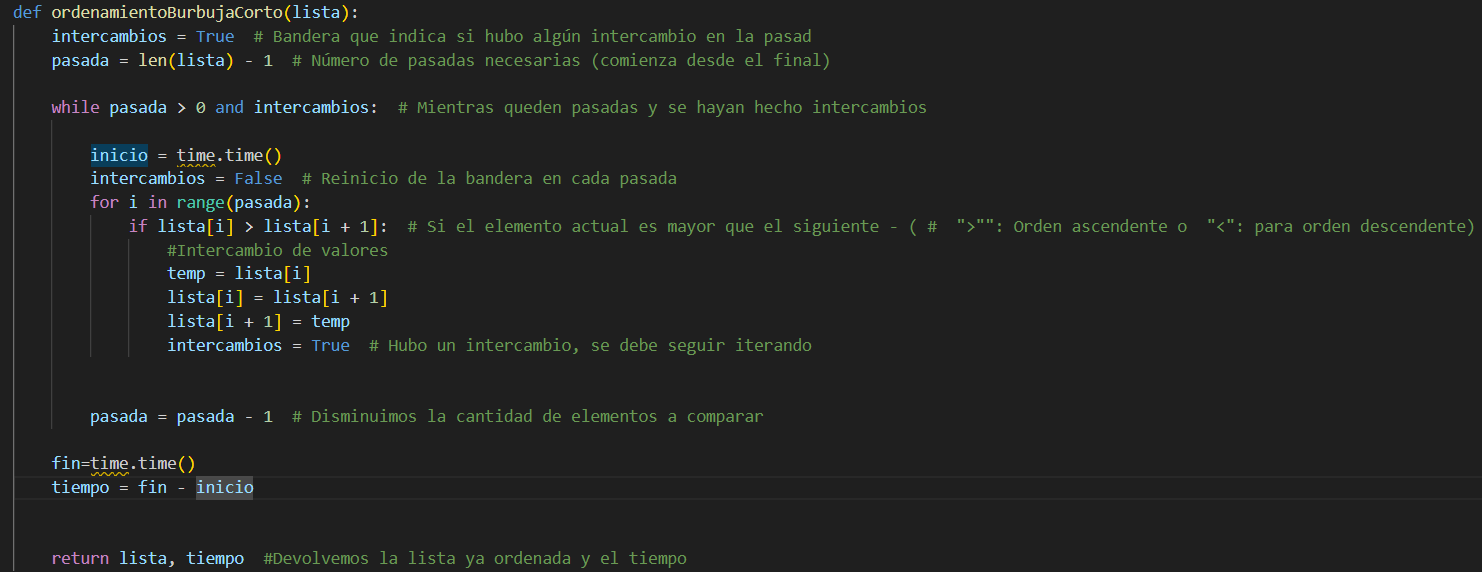
Lista 1:



Lista 2:

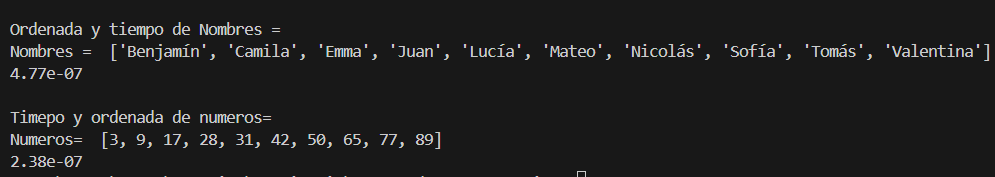


Método de ordenamiento que aplicaremos:

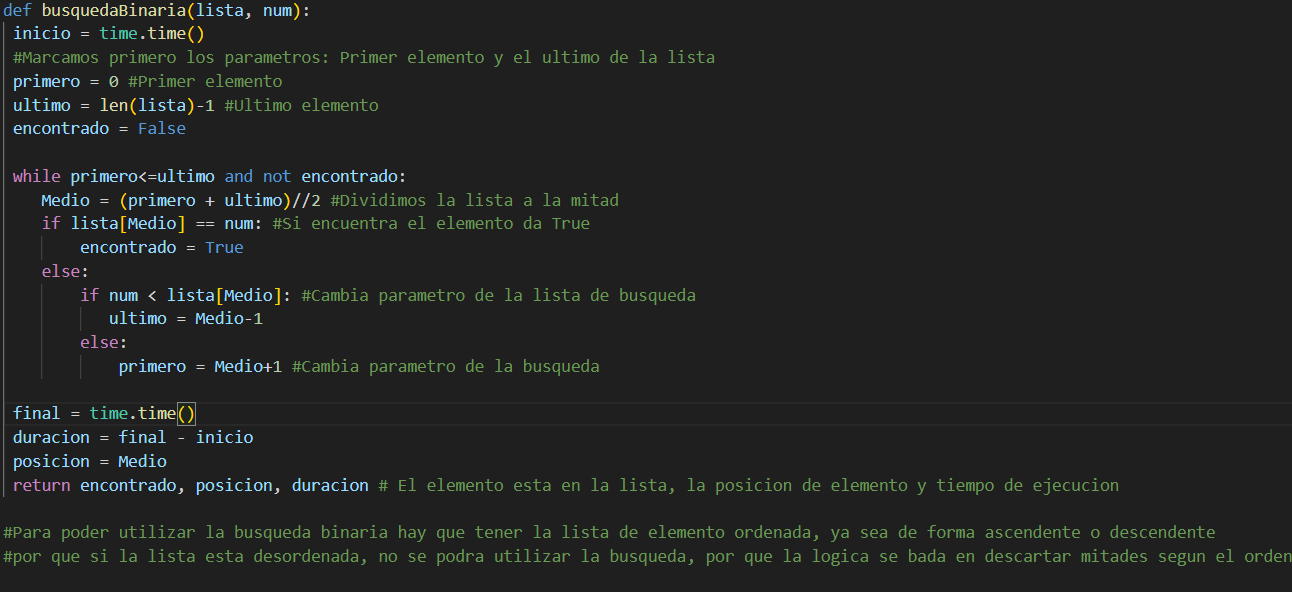


En donde como parte final nos dará como resultado el tiempo que demoro en ordenar la lista y la lista ordenada.

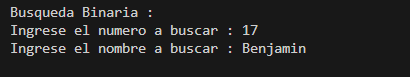
Como vemos en la captura, nos muestras las listas ordenadas.

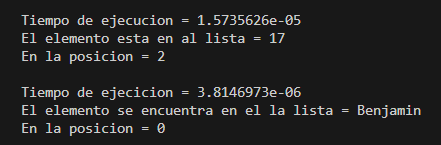


Ahora vamos a aplicar la búsqueda binaria:



Pedimos los datos de los elementos que queremos saber de las listas:





Vemos en terminal los resultados que nos dio, la búsqueda binaria: El tiempo que demoro en buscar el elemento y la posición del elemento en la lista

**Metodología Utilizada**

El desarrollo del trabajo se llevó a cabo siguiendo las siguientes etapas:

**Recolección de información:** Se consultaron fuentes confiables para comprender los fundamentos.

**Implementación en Python:** Se programaron los algoritmos en lenguaje Python para facilitar su prueba y análisis.

**Pruebas con datos variados:** Se realizaron ensayos con distintos conjuntos de datos para verificar el comportamiento de cada algoritmo. (Integer y Strings)

**Registro y validación:** Se documentaron los resultados obtenidos y se validó la correcta funcionalidad de las soluciones desarrolladas.

**Redacción del informe:** Finalmente, se elaboró el presente informe acompañado de los anexos correspondientes.

**Resultados obtenidos**

Los algoritmos funcionaron correctamente. Se observaron los siguientes resultados:

* Las listas se ordenaron exitosamente con **Bubble Sort optimizado**.
* La **búsqueda binaria** identificó correctamente los elementos cuando estaban presentes.
* Se registraron los tiempos de ejecución para ambos procesos, que variaron según el tamaño de las listas y el caso.

**Conclusiones**

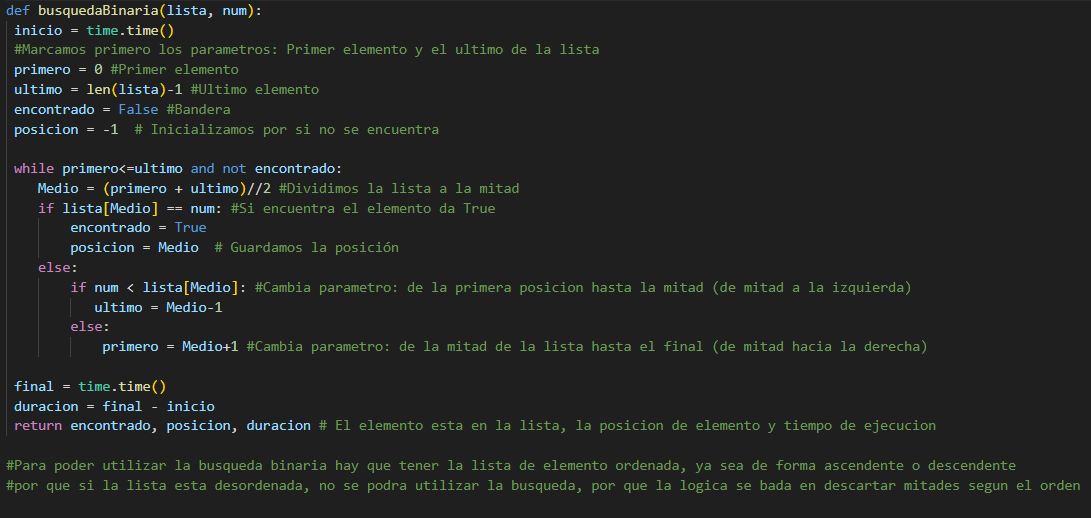
* El ordenamiento previo de la lista es fundamental para utilizar búsqueda binaria.
* La versión optimizada del algoritmo burbuja mejora el rendimiento cuando la lista está parcialmente ordenada.
* Python permite medir el tiempo de ejecución de forma sencilla, lo cual es útil para análisis de eficiencia.
* La búsqueda binaria demostró ser significativamente más rápida que una búsqueda secuencial en listas grandes.
* Es importante validar y limpiar los datos (como eliminar tildes) para evitar errores en la ejecución.

**Bibliografía**

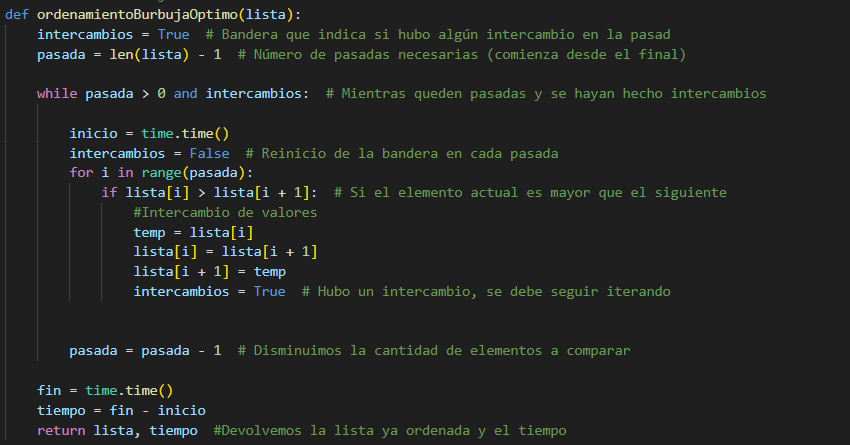
* Problem Solving with Algorithms and Data Structures using Python by Bradley N. Miller, David L. Ranum is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/). [**https://runestone.academy/ns/books/published/pythoned/index.html**](https://runestone.academy/ns/books/published/pythoned/index.html)
* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
* Python Software Foundation. <https://docs.python.org/3/>
* GeeksforGeeks. Data Structures and Algorithms. <https://www.geeksforgeeks.org>

**Anexos**

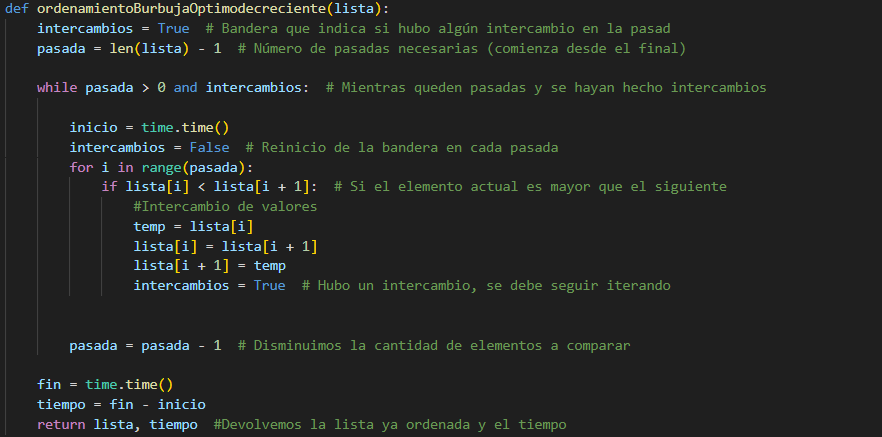
Búsqueda Binaria:

****

Ordenamiento Burbuja Optimo: Creciente

****

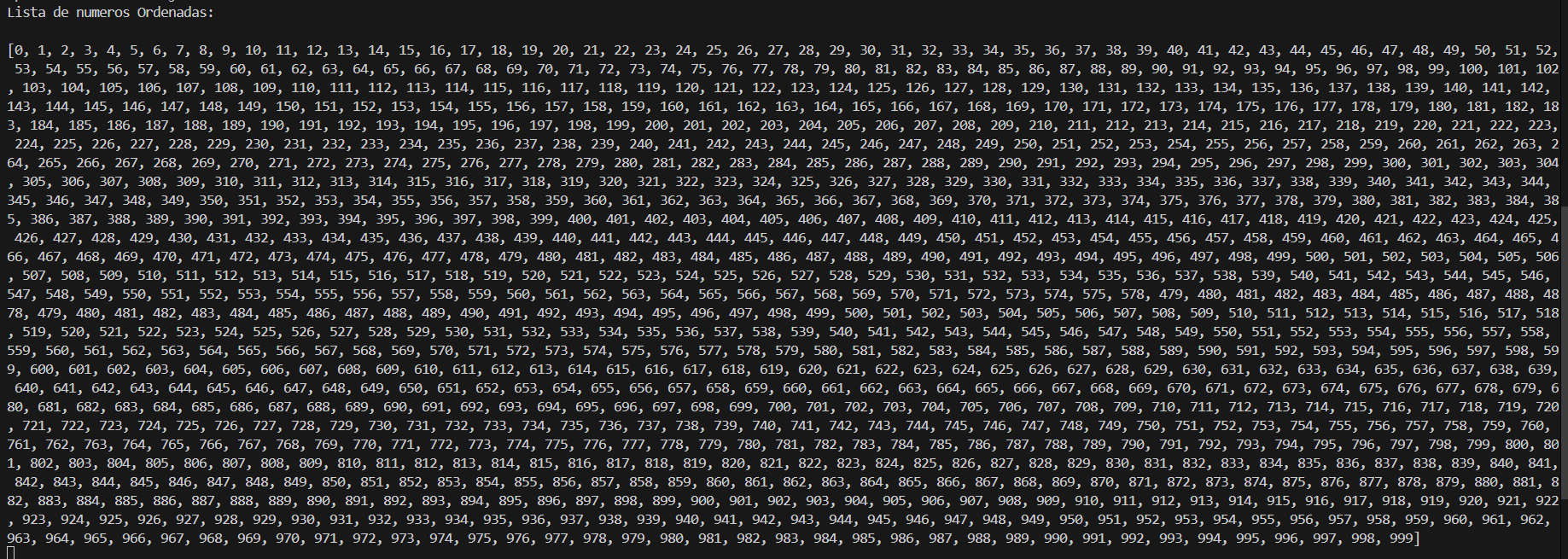
Ordenamiento Burbuja Optimo: Decreciente



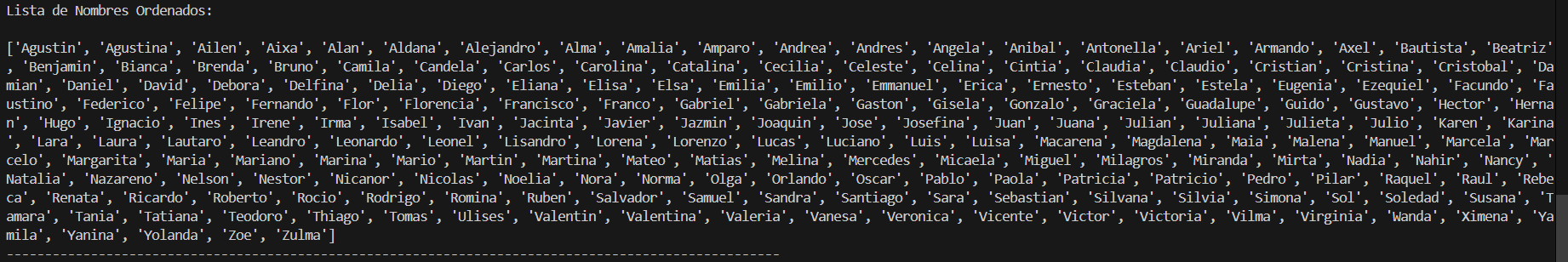
Listas Utilizadas:

****

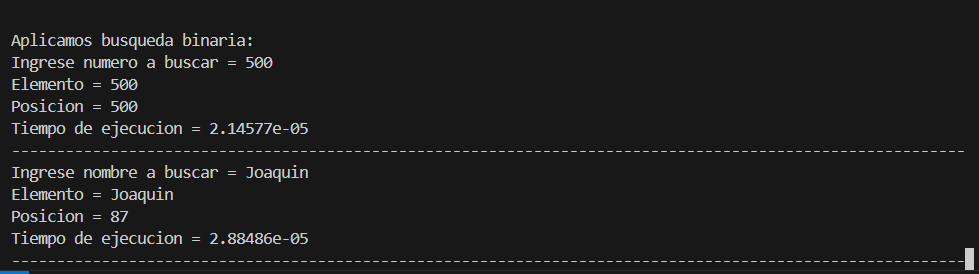
**Lista de números ordenadas**

****

**Lista de nombres ordenada:**

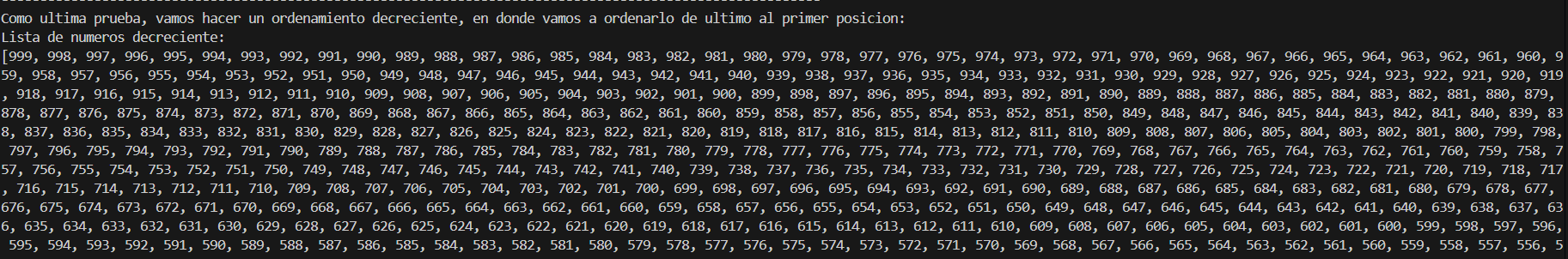
****

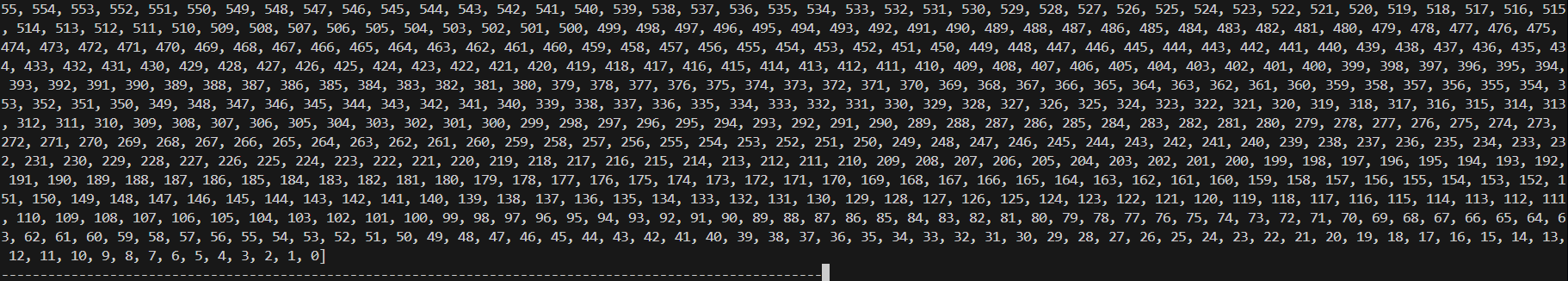
**Aplicamos búsqueda binaria:**

****

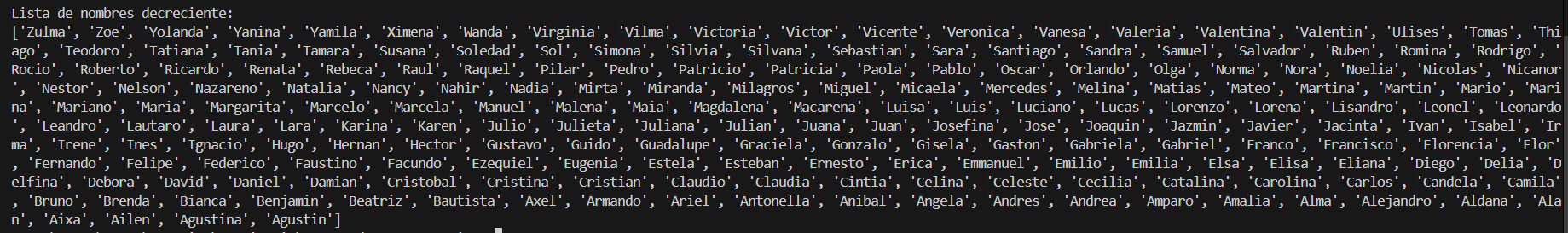
**Aplicamos Ordenamiento de Listas Decrecientes:**

**Números:** Desde 999 a 0.

****

****

**Nombres:** Desde Z hasta la A.

****