****

**Trabajo Práctico Integrador Programación I**

**Materia: Programación I**

**Carrera: Tecnicatura Universitaria en Programación**

**Institución: Universidad Tecnológica Nacional (UTN)**

**Fecha de Entrega: 09 de Junio**

**Integrantes:**

**Nazareno Malpassi - nazarenomalpassi@gmail.com  
Francisco López -** [**franciscolodev@gmail.com**](mailto:franciscolodev@gmail.com)

**Introducción**

Este trabajo práctico integrador fue realizado por los estudiantes Francisco López y Nazareno Malpassi, en el marco de la materia Programación I, con el objetivo de aplicar y profundizar los conocimientos adquiridos durante el cursado. En esta oportunidad

Nos enfocaremos en dos tareas fundamentales en el manejo de información: el ordenamiento de datos y la búsqueda de datos específicos. Para ilustrar estos conceptos de manera clara y práctica, hemos desarrollado un programa en Python que simula una situación cotidiana: la gestión de notas de alumnos. Utilizaremos el algoritmo de Ordenamiento por Selección para organizar las notas de la clase y, posteriormente, aplicaremos el método de Búsqueda Binaria para encontrar si una nota en particular existe dentro de nuestro registro ordenado. Este enfoque nos permitirá comprender no solo cómo funcionan estas metodologías, sino también la importancia de la organización de los datos para realizar búsquedas de manera más eficiente.

A través de este trabajo, no solo buscamos repasar conceptos teóricos, sino también llevarlos a la práctica mediante la implementación de distintos algoritmos utilizando el lenguaje Python, explorando casos concretos que nos permitan visualizar su funcionamiento. Creemos que esta experiencia es clave para afianzar los conocimientos y prepararnos mejor para desafíos más avanzados en nuestra formación como programadores.

**Marco Teórico y Metodologías Utilizadas**

¿Qué son los Algoritmos y Por Qué los Usamos?

En programación, un algoritmo es como una serie de instrucciones paso a paso para resolver un problema. Imagina que es una "receta" para tu computadora. Aquí vamos a ver recetas para dos problemas comunes con listas de datos (como nuestras notas de alumnos):

1. Ordenar: Poner los datos en un orden específico (por ejemplo, de menor a mayor).
2. Buscar: Encontrar si un dato en particular existe en la lista y dónde está.

1. El Algoritmo para Ordenar: Selection Sort (Ordenamiento por Selección)

* La Idea Principal: Este algoritmo es muy intuitivo. Para ordenar una lista de notas de menor a mayor, el Selection Sort hace lo siguiente:
  + Busca la nota más baja de toda la lista y la coloca en la primera posición.
  + Luego, busca la nota más baja de la parte que queda (sin contar la primera posición) y la coloca en la segunda posición.
  + Repite este proceso, "seleccionando" la nota más baja de lo que resta y poniéndola en su lugar, hasta que toda la lista esté ordenada.
* ¿Cómo se ve esto en el código?
  + La función ordenar\_notas\_por\_seleccion() hace esto.
  + El código usa un bucle principal que va revisando cada posición.
  + Dentro de este bucle, hay otro bucle más pequeño que busca cuál es el valor más bajo en la parte restante de la lista y guarda su índice (su posición).
  + Al final de cada pasada, se hace un intercambio: el valor que estaba en la posición actual se cambia por el valor más bajo que encontramos. Es decir, lista\_notas[i], lista\_notas[indice\_minimo] = lista\_notas[indice\_minimo], lista\_notas[i].

2. El Algoritmo para Buscar: Binary Search (Búsqueda Binaria)

* La Idea Principal (Necesita una lista ordenada): Esta búsqueda es muy inteligente, pero solo funciona si tu lista de datos (nuestras notas) ya está ordenada. Es como buscar una palabra en un diccionario:
  1. Abrís el diccionario justo por la mitad.
  2. Mirás la palabra de esa página del medio.
  3. Si la palabra que buscás es alfabéticamente anterior a la del medio, ¡sabés que solo tenés que buscar en la primera mitad!
  4. Si es posterior, entonces buscás en la segunda mitad.
  5. Seguís repitiendo esto: ¡dividir la parte que te queda por la mitad! Así, cada vez que mirás, descartás la mitad de las posibilidades.
* ¿Cómo se ve esto en el código?
  1. La función buscar\_nota\_binario() implementa esta lógica.
  2. Primero, tenemos un izquierda y un derecha que marcan los límites de la parte de la lista que estamos revisando.
  3. Mientras izquierda no pase a derecha, seguimos buscando:
     + Calculamos el medio del rango actual.
     + Comparamos la nota\_buscada con la nota que está en la posición medio.
     + Si son iguales, ¡la encontramos! Devolvemos su posición (índice).
     + Si la nota de medio es menor que la nota\_buscada, sabemos que nuestra nota está en la mitad de la derecha (donde están las notas más grandes), así que ajustamos izquierda para que empiece un lugar después del medio.
     + Si la nota de medio es mayor, sabemos que nuestra nota está en la mitad de la izquierda (donde están las notas más chicas), así que ajustamos derecha para que termine un lugar antes del medio.
  4. Si el proceso termina y no la encontramos, devolvemos -1.

¿Por Qué Importa Ordenar Antes de Buscar con Búsqueda Binaria?

Nuestro programa muestra cómo funcionan estos algoritmos. Si mirás la parte de la búsqueda, notarás que la buscar\_nota\_binario usa la lista notas\_ordenadas. Esto es clave: la Búsqueda Binaria es increíblemente eficiente (rápida) precisamente porque puede descartar la mitad de las opciones en cada paso, ¡pero solo si los datos están organizados! Si no estuvieran ordenados, tendríamos que buscar nota por nota, lo cual es mucho más lento en listas grandes.

Este ejemplo simple te ayuda a ver cómo la organización de los datos (el ordenamiento) es un paso muy útil para poder hacer búsquedas muy rápidas después.

En la sección ANEXO, estaremos poniendo las imágenes del código que realizamos para el caso práctico.

**Resultados obtenidos**

Al ejecutar el programa desarrollado, pudimos observar de forma directa el comportamiento y la efectividad de los algoritmos de ordenamiento y búsqueda implementados. A continuación, se detallan los resultados clave:

1. **Ordenamiento de Notas con "Selection Sort":**
   * Inicialmente, el programa genera una lista de 12 notas aleatorias, simulando un escenario desorganizado.
   * Tras aplicar el algoritmo de **Ordenamiento por Selección**, la lista de notas se transforma eficientemente, presentando todos los valores en **orden ascendente (de menor a mayor)**. Este proceso ilustra cómo los elementos son seleccionados y reubicados en su posición correcta de manera secuencial.
2. **Búsqueda de Notas con "Binary Search":**
   * Una vez que las notas están ordenadas, el programa demuestra la funcionalidad de la **Búsqueda Binaria**.
   * Al buscar una nota que **sabemos que existe** dentro de la lista ordenada, el algoritmo la localiza exitosamente, retornando su **índice (posición)** en la lista.
   * Por otro lado, al buscar una nota que **sabemos que no existe** en la lista (por ejemplo, un valor fuera del rango esperado), el algoritmo concluye correctamente que el elemento **no se encuentra**, indicándolo con un valor de -1.

En síntesis, los resultados obtenidos confirman que el programa cumple su propósito didáctico al mostrar cómo un algoritmo de ordenamiento (Selection Sort) organiza los datos, y cómo esta organización es fundamental para que un algoritmo de búsqueda altamente eficiente (Binary Search) pueda encontrar elementos de manera rápida y precisa.

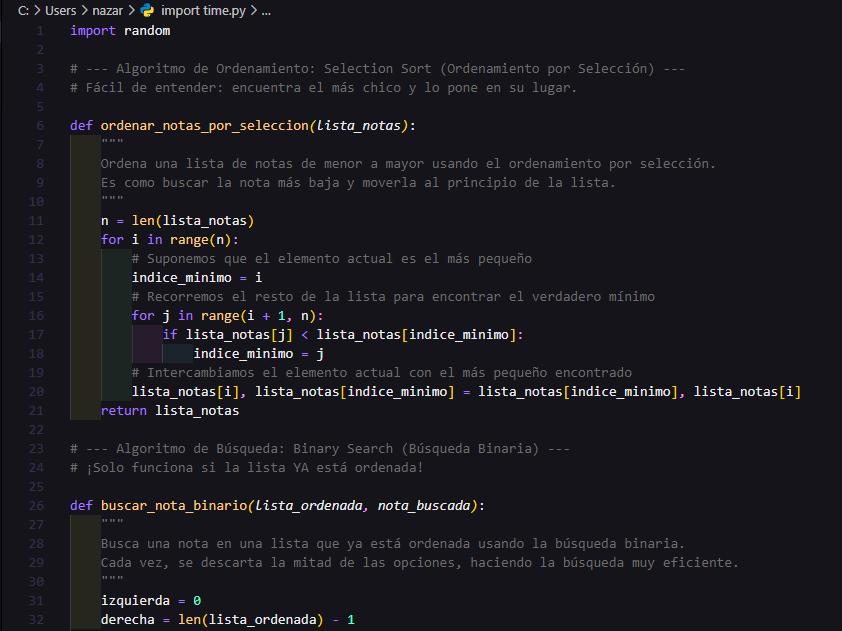
**Conclusión**

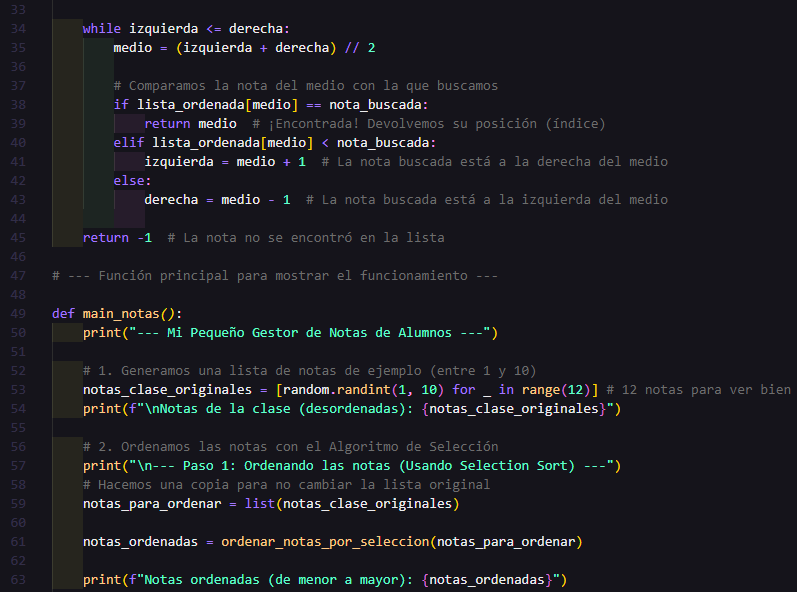
En el transcurso de este trabajo práctico, hemos abordado los conceptos fundamentales de los algoritmos de ordenamiento y búsqueda, demostrando su aplicación en un escenario práctico de gestión de notas estudiantiles mediante el lenguaje de programación Python. Se implementó el Ordenamiento por Selección, lo que nos permitió comprender el proceso iterativo de ubicar elementos en su posición correcta dentro de una secuencia.

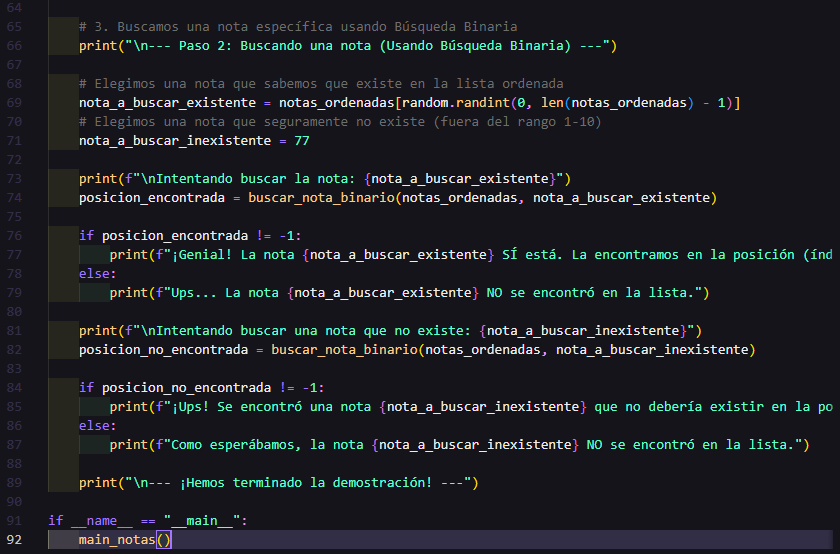
Posteriormente, la utilización de la Búsqueda Binaria puso de manifiesto la crítica dependencia entre la organización de los datos y la eficiencia de las operaciones de búsqueda. Se evidenció que, al operar sobre una colección de datos previamente ordenada, este algoritmo optimiza significativamente el tiempo de localización de un elemento, al descartar sistemáticamente la mitad del espacio de búsqueda en cada iteración.

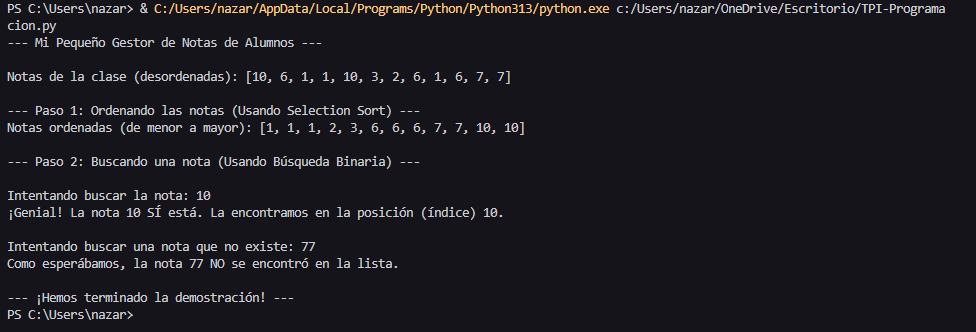
En síntesis, esta experiencia práctica no solo facilitó la comprensión de la lógica inherente a estos algoritmos esenciales, sino que también subrayó la trascendental importancia de la estructuración de los datos para la optimización del rendimiento algorítmico. La selección apropiada de un algoritmo constituye un factor determinante en la eficiencia y escalabilidad de las soluciones de software, especialmente al procesar volúmenes considerables de información.

**Anexo**









**Bibliografía**

-“Apuntes de clase – Programación I – Tecnicatura Universitaria en Programación (UTN).”  
-“Teoría sobre Búsqueda y Ordenamiento en Programación”

-“Notebook Búsqueda y Ordenamiento”

-“Python Software Foundation. (2024). Python 3.12 Documentation”. [(https://docs.python.org/3/](https://docs.python.org/3/))

**-"Introducción a los Algoritmos"** de Thomas H. Cormen, Charles E. Leiserson, Ronald L.

**-"Algoritmos y Estructuras de Datos"** de Niklaus Wirth.