Universidad Nacional de Entre Ríos

➢*Facultad:* Facultad de Ingeniería.

➢*Carrera:* Licenciatura en Bioinformática y Bioingeniería.

➢*Cátedra:* Algoritmos y estructuras de datos

➢*Docentes:* Javier E. Diaz Zamboni, Jordán F. Insfrán, Juan F. Rizzato

➢*Título del trabajo:* "Aplicaciones de TADs- "*Problema 1 ""*.

➢*Alumnos:*

- Almirón Spahn, María Paz
- Leiva, Giuliana
- Saravia, Lucía Milagros

➢*Fecha de entrega:* 26 de Septiembre del 2025

Problema 1:

En la presente actividad creamos una Lista Doblemente Enlazada inicialmente vacía, donde implementamos distintas especificaciones lógicas, tales como:

- **esta_vacia()**: Devuelve **True** si la lista está vacía.
- **agregar_al_inicio(ítem)**: Agrega un nuevo ítem al inicio de la lista.
- **agregar_al_final(ítem)**: Agrega un nuevo ítem al final de la lista.
- **insertar(item, posición)**: Agrega un nuevo ítem a la lista en "posición". Si la posición no se pasa como argumento, el ítem debe añadirse al final de la lista. Además, si la posición no es válida se arroja una excepción.
- **extraer(posición):** elimina y devuelve el ítem en "posición". Si no se indica el parámetro posición, se elimina y devuelve el último elemento de la lista. Además se agrega la excepción pedida por la cátedra.
- **copiar():** Realiza una copia de la lista elemento a elemento y devuelve la copia
- **invertir():** Invierte el orden de los elementos de la lista.
- **concatenar(Lista):** Recibe una lista como argumento y retorna la lista actual con la lista pasada como parámetro concatenada al final de la primera.
- **__len__()**: Devuelve el número de ítems de la lista.
- **__add__(Lista)**: El resultado de “sumar” dos listas debería ser una nueva lista con los elementos de la primera lista y los de la segunda. Aprovechar el método concatenar para evitar repetir código.
- **__iter__()**: permite que la lista sea recorrida con un ciclo for

De la misma manera, se midieron los tiempos de ejecución de tres métodos implementados en la clase “ListaDobleEnlazada”, estos fueron:

- ***len():*** Devuelve el número de ítems de la lista.
- ***copiar():*** Realiza una copia de la lista elemento a elemento y devuelve la copia.
- ***invertir():*** Invierte el orden de los elementos de la lista.

A Priori de los resultados:

Al observar el código, es posible predecir que el método len() tendrá un orden de complejidad O(1), esto se debe a que el tamaño no influye en el tiempo de ejecución. En cambio, los otros dos métodos serán de orden O(n), debido a que recorren uno a uno los elementos de la lista, lo cual se contempla en el código al utilizar “while”.

Resultados de las gráficas:

La representación gráfica muestra cómo varía el tiempo de ejecución en función del tamaño de la lista.

- ***len():*** Se observa como la función se mantiene prácticamente constante al aumentar el número de elementos, lo que confirma que la complejidad es **O(1)**.
- ***copiar():*** incrementa su tiempo de ejecución de manera lineal a medida que crece el tamaño de la lista, lo que corresponde a **O(n).**
- ***invertir():*** presenta un comportamiento similar a copiar(), ya que también requiere recorrer la lista completa, por lo que su orden de complejidad es **O(n)**.

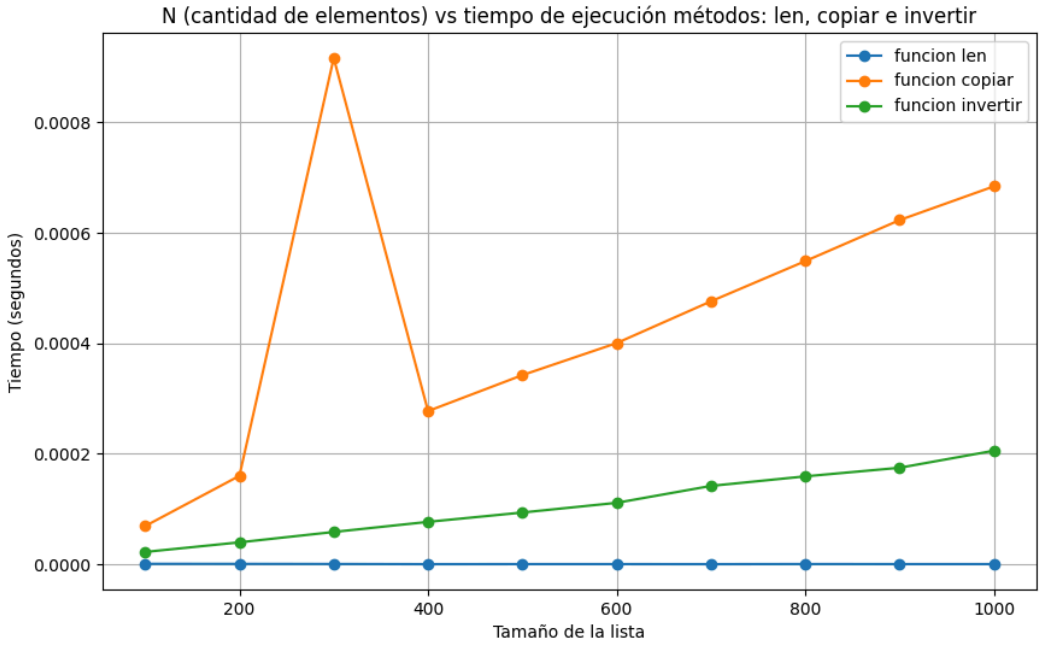

Figura 1: N (cantidad de elementos) vs tiempo de ejecución de los métodos: len, copiar e invertir

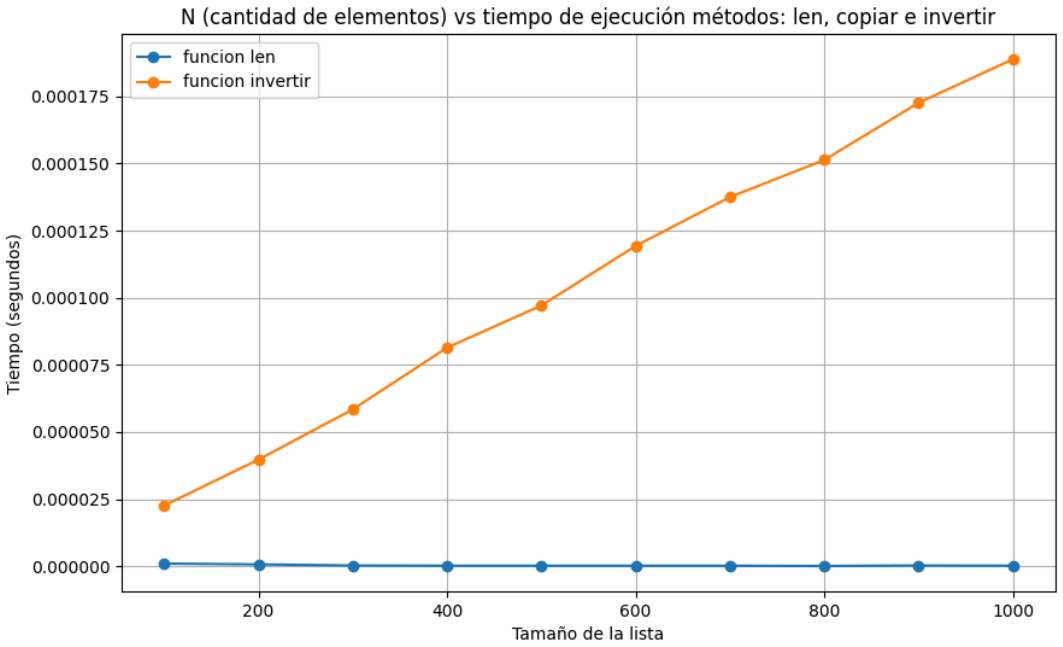

Figura 2: N (cantidad de elementos) vs tiempo de ejecución de los métodos: len e invertir

Para concluir, es posible establecer que se cumple con el orden de complejidad de los métodos pedidos. Asimismo, el código logró superar satisfactoriamente los diferentes test proporcionados por la cátedra, lo que nos permite verificar el correcto funcionamiento de los métodos. Esto no quiere decir que el código no tiene falla alguna sino que los posibles errores, ya considerados por la cátedra, fueron superados con éxito.