PROBLEMA 2

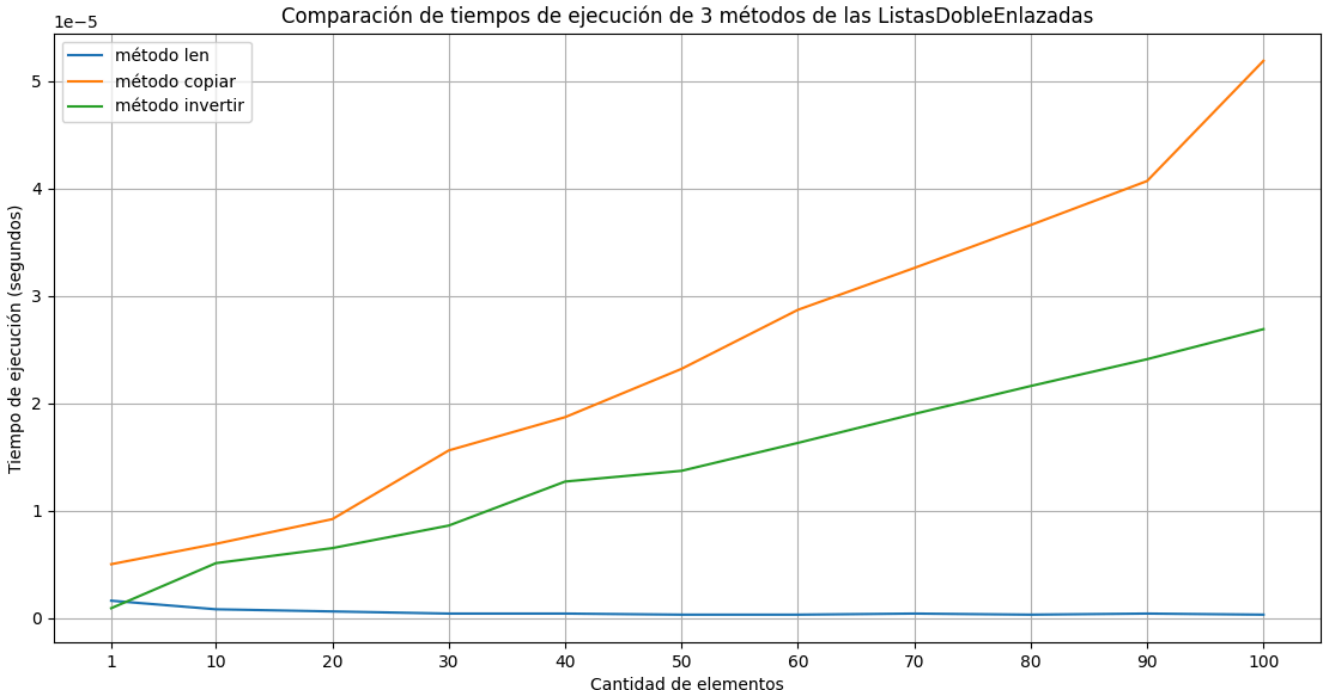

Figura 1: Gráfico de cantidad de elementos vs. Tiempo de ejecución.

A partir del código creado, podemos deducir que:

- El método "len" es el que se ejecuta con mayor rapidez, teniendo un orden de complejidad constante (O(1)).

- El método "invertir" se ejecuta más lento que el método len, pero más rápido que el método copiar:
    - Las primeras líneas verifican si el tamaño de la lista es menor o igual a 1. Estas operaciones y el posible retorno tienen un costo constante, O(1).
    - Luego, se intercambian la cabeza y la cola de la lista (O(1)).
    - El bucle while actual is not None se ejecuta n veces, donde n es el número de elementos en la lista.
    - Dentro del bucle, se intercambian los punteros siguiente y anterior del nodo actual (O(1)).
    - Finalmente, se retorna la propia lista (self), que ahora está invertida (O(1)).

  Por lo tanto, el **orden de complejidad del método** invertir(self) es O(n), ya que el bucle while itera sobre cada elemento de la lista una sola vez, realizando operaciones de tiempo constante en cada iteración.

- Por último, el método "copiar" es el que se ejecuta más lento:
    - El método inicializa una nueva lista doblemente enlazada (lista _copia). Esta operación tiene un costo constante, O(1).
    - Luego, itera a través de cada nodo de la lista original utilizando un bucle while que se ejecuta hasta que actual sea None. En el peor caso, esto ocurrirá n veces, donde n es el número de elementos en la lista original.
    - Como agregar_al_final tiene una complejidad constante O(1) y se llama n veces dentro del bucle, la complejidad total del bucle es O(n) * O(1) = O(n).

Por lo tanto, el **orden de complejidad del método** copiar(self) es O(n), lo que significa que el tiempo de ejecución crece linealmente con el número de elementos en la lista.