

TP N°1

Algoritmos y Estructuras de Datos

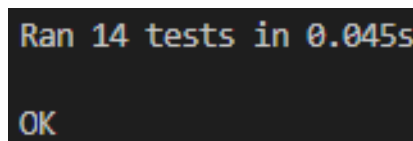
Deleon, Iván; Gerbaudo, Sabina; Rogau, Victoria.

Problema 1

En el problema N°1 se implementó el TAD lista doblemente enlazada como clase de Python con las distintas funciones, respetando la siguiente especificación lógica:

- **esta_vacia()**: Devuelve **True** si la lista está vacía.
- **agregar_al_inicio(item)**: Agrega un nuevo ítem al inicio de la lista.
- **agregar_al_final(item)**: Agrega un nuevo ítem al final de la lista.
- **insertar(item, posicion)**: Agrega un nuevo ítem a la lista en "posicion". Si la posición no se pasa como argumento, el ítem debe añadirse al final de la lista. "posicion" es un entero que indica la posición en la lista donde se va a insertar el nuevo elemento. Si se quiere insertar en una posición inválida, que se arroje la debida excepción.
- **extraer(posicion)**: elimina y devuelve el ítem en "posición". Si no se indica el parámetro posición, se elimina y devuelve el último elemento de la lista. La complejidad de extraer elementos de los extremos de la lista debe ser $O(1)$. Si se quiere extraer de una posición indebida, que se arroje la debida excepción.
- **copiar()**: Realiza una copia de la lista elemento a elemento y devuelve la copia. Verificar que el orden de complejidad de este método sea $O(n)$ y no $O(n^2)$.
- **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

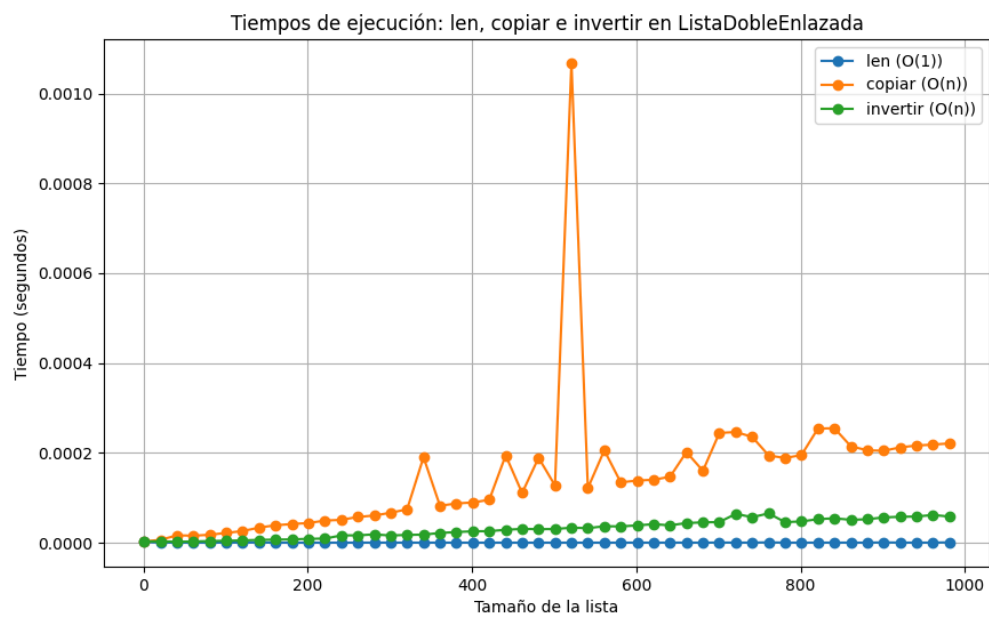
Dicho código pasó las pruebas unitarias provistas por la cátedra de AyEDs.



```
Ran 14 tests in 0.045s
OK
```

Comprobación del test para lista doblemente enlazada.

Posteriormente se realizaron gráficas de cantidad de elementos (n) vs tiempo de ejecución para los métodos `__len__()`, `invertir()` y `copiar()` con las cuales pudimos comprobar su eficiencia temporal. En la función `__len__()` se observa una complejidad de $O(1)$. Por otro lado, en las funciones `invertir()` y `copiar()` obtenemos $O(n)$, esto se corresponde con el resultado esperado, pues copiar e invertir requieren recorrer los n elementos de la lista.



Gráfica de las funciones __len__, copiar e invertir