# Trabajo Práctico N°1

**Alumno:** Pucheta Luciano Valentin

**Materia:** Algoritmo y Estructura de Datos

**Docentes:** Javier Eduardo Diaz Zamboni, Jordán F. Insfrán y Bruno M. Breggia

# Ejercicio 1: Lista Doblemente Enlazada.

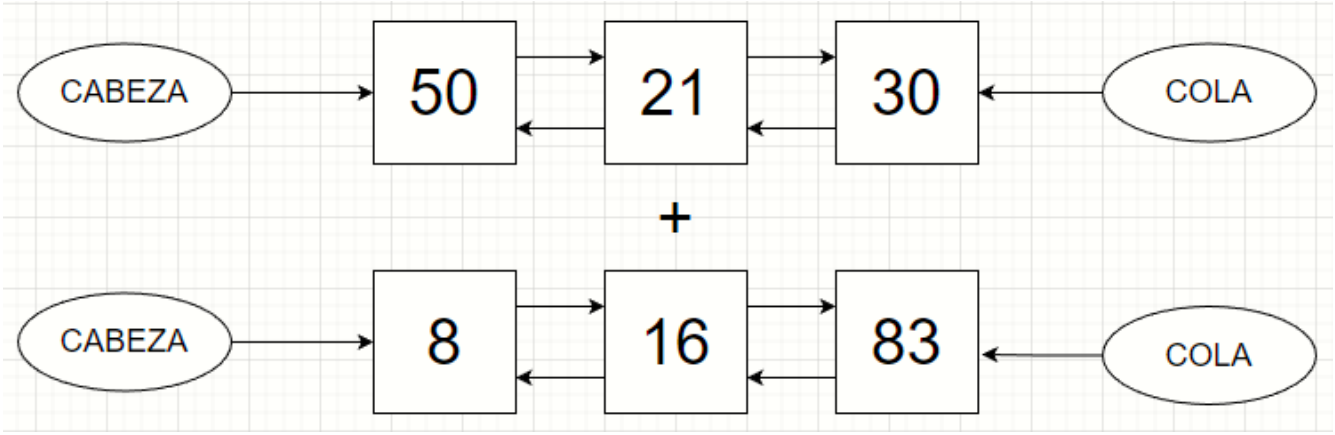
A continuación  explico algunos métodos de la clase y su orden de complejidad.

**Atributos del init :**

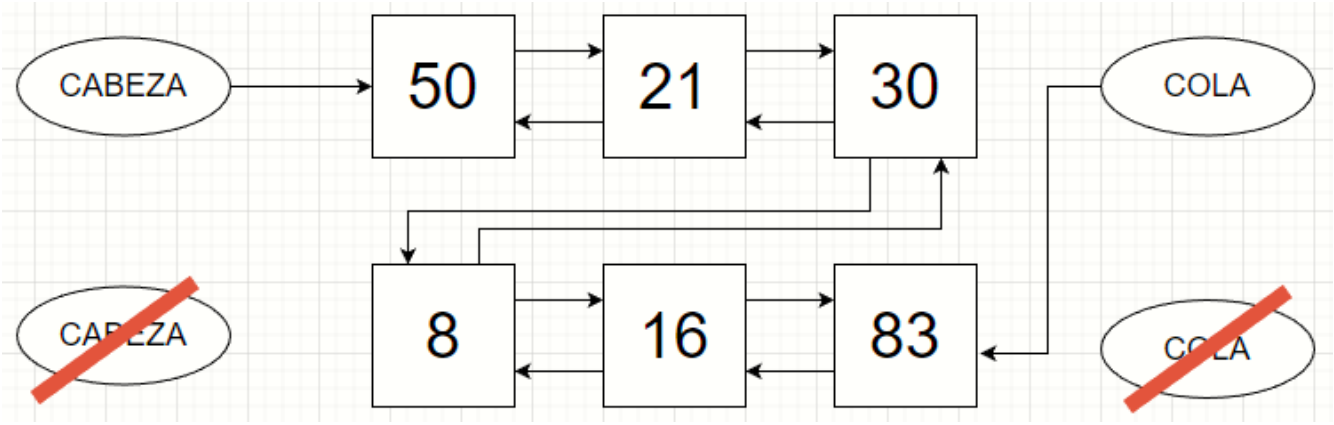
- Tamaño: registra el numero de nodos dentro de la lista
- Cabeza: registra el primer nodo de la lista
- Cola: registra el ultimo nodo en la lista

**Método de concatenación $O(n)$ :**

Primeramente, se crean 2 copias, una de la lista original, y otra de la que se planea concatenar a la primera, esto a fin de no modificar las listas originales en el proceso.




La lista resultante, se almacena en la variable asignada a la copia de la lista original, por lo que prescindimos tanto de la cola como la cabeza de la segunda lista, y enlazamos la cola de la primera con la cabeza de la segunda, seguido de esto, la nueva cola de la primera lista pasa a ser la misma que la cola de la segunda lista.

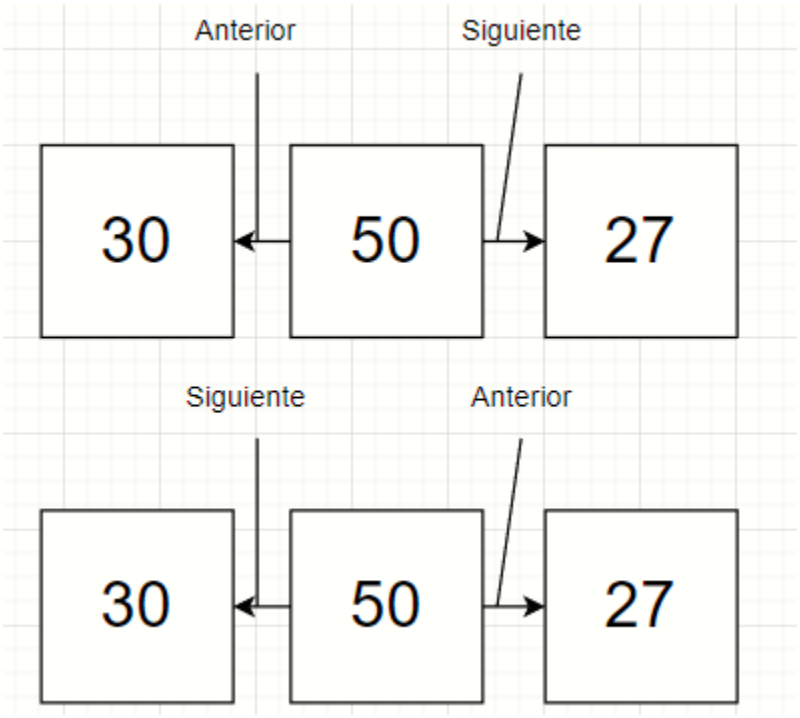


Dado que el método concatenar utiliza internamente el método copiar, y siendo este de orden de complejidad O(n). El orden de complejidad total del método concatenar es O(n)

**Método invertir** O(n) **:**

El método invertir recorre secuencialmente la lista, invirtiendo las referencias de anterior y siguiente para cada nodo. Dado que el número de iteraciones es igual a la cantidad de nodos en la lista, el orden de complejidad del algoritmo es O(n).

Entonces, para cada nodo dentro de la lista:




**Algoritmo de ordenamiento O($n^2$):**

La clase Lista Doblemente enlazada se ordena por medio de el algoritmo de ordenamiento por inserción. El gráfico a continuación muestra los tiempos de ejecución del algoritmo de 100 hasta 5000 elementos, con un incremento de 100 entre cada medición (el código con el que se generó el gráfico se encuentra en el módulo de “main.py” del repositorio).


Análisis del algoritmo:

```
def ordenar(self):
    """ Ordena la lista utilizando el algoritmo de ordenamiento insercion"""


    #si la lista esta vacia entonces ya esta ordenada
    if self.esta_vacia():                                          # O(1)
        return


    nodo_actual =self.cabeza.siguiente                             # O(1)
    while nodo_actual is not None:                                 # O(n)
        dato = nodo_actual.dato                                    # O(n)
        nodo_comparar = nodo_actual.anterior                       # O(n)

        while nodo_comparar is not None and nodo_comparar.dato > dato:      # O(n**2)
            nodo_comparar.siguiente.dato = nodo_comparar.dato       # O(n**2)
            nodo_comparar = nodo_comparar.anterior                 # O(n**2)

        if nodo_comparar is None:                                  # O(n)
            self.cabeza.dato = dato                                # O(n)
        else:
            nodo_comparar.siguiente.dato = dato                    # O(n)

        nodo_actual = nodo_actual.siguiente                        # O(n)
```

*Figura 1: Linea 145 del módulo "ListaDobleEnlazada.py"*

Todo lo que esta por fuera del primer bucle while, se ejecuta una sola vez, por lo que su tiempo de ejecución es constante O(1), por dentro del primer bucle cada linea se ejecuta N veces O(n), y cada linea dentro del segundo bucle se ejecuta N veces también, pero al ser un bucle anidado

ese while también se ejecuta N veces, por lo que cada linea dentro de el se ejecuta N*N o bien $O(n^2)$.

La formula general para este caso quedaría de la siguiente manera:

$$3\,n^2 + 7n + 2$$

siendo que $3\,n^2$ es el miembro mas representativo, el tiempo de ejecución del algoritmo es **$O(n^2)$.**

# Ejercicio 2: Juego De Guerra.

El módulo “guerra.py” contiene 3 clases:

- **Carta:** Modela, como su nombre lo indica, cada carta del juego, posee los atributos palo, valor, y boca arriba, ademas de métodos para mostrar las cartas y compararlas.
- **Cola Doble:** es una estructura de cola doble que se usa para modelar el mazo de cartas de cada jugador.
- **Juego de Guerra:** contiene los mazos de los jugadores, registra la cantidad de turnos y posee todos los métodos propios del juego como el duelo o la guerra. Ademas tiene 3 atributos que se utilizan como auxiliares para mostrar el progreso del juego por consola.

**Método duelo O(1):**

Este método es el primero y el que mas se ejecuta, se ocupa tomar una carta del mazo de cada jugador, compararlas, repartirlas al ganador, corroborar que cada jugador tiene cartas suficientes para continuar y llamar al método guerra si ocurriese un empate.
Para comparar las cartas se utiliza el atributo “valor” de la clase carta, salvo en los casos donde no hay un valor numérico almacenado en dicho atributo. En dado caso, los puntos para comparar se calculan mediante la siguiente función:

```
def puntos (valor1):
    """Calcula los puntos a una carta para su comparacion"""

    if valor1 in ['A','J','Q','K']:
        if valor1 =='A':
            valor1=14
        elif valor1== 'J':
            valor1=11
        elif valor1== 'Q':
            valor1=12
        elif valor1=='K':
            valor1=13

    else:
        valor1=int(valor1)

    return valor1
```

*Figura 2: Linea 8 del módulo "guerra.py"*

Si el ganador del juego se define en un duelo, la función retorna “1” o ”2”  dependiendo del ganador, o bien retorna un string vacío si el juego continua.

**Método Guerra:**

El método “guerra” es llamado por el método “duelo” cuando ocurre un empate al realizar la comparación. Se encarga de ejecutar la mecánica de guerra, corroborando que ambos jugadores tengan cartas suficientes para seguir y repartir las cartas disputadas al ganador.
Si llega a darse el caso de que al sacar las ultimas cartas para comparar, ocurriese otro empate, se llamará recursivamente al método guerra, hasta tener un ganador.

# Ejercicio 3: Ordenar archivo de texto

Tomando el archivo “datos.txt” generado a partir del código proporcionado por la cátedra, se implementaron las siguientes funciones dentro del módulo “ordenamiento_externo.py”

**Generar_archivos:** Recibe como parámetro el nombre del archivo a dividir y un valor B que representa el tamaño del bloque en el que se dividirá el archivo.

**Ordenar_archivos:** Ordena individualmente cada bloque generado

**Mezcla_directa_archivos:** Recibe como parámetro el nombre de 2 archivos y el nombre que se desea para el archivo resultante. Ordena el contenido de los 2 archivos por medio de mezcla directa y almacena el resultado en un archivo nuevo, posteriormente elimina los archivos que tomo para ordenar.

**Mezclar_todo:** Recibe como parámetro el numero total de bloques a mezclar y comienza a mezclar archivos de a pares hasta que solo quede uno.