Algoritmos y Estructuras de Datos

UNER Facultad de Ingeniería

Trabajo Práctico N° 1

Integrantes: Gabirondo Emiliano y Lambarri Joaquín

Análisis del orden de complejidad del algoritmo de ordenamiento

El algoritmo de ordenamiento utilizado en el ejercicio 1 fue el ordenamiento burbuja.
Si tenemos una lista de tamaño n, la cantidad de pasadas necesarias para el ordenamiento de la misma será n-1, para el peor de los casos.

```python
def ordenar(self):
    """
    Ordena los elementos de la lista de menor a mayor.
    """
    if self.tamanio <= 1:
        return self

    nodo_actual = self.cabeza
    while nodo_actual is not None:
        nodo_siguiente = nodo_actual.siguiente
        while nodo_siguiente is not None:
            if nodo_actual.dato > nodo_siguiente.dato:
                nodo_actual.dato, nodo_siguiente.dato = nodo_siguiente.dato, nodo_actual.dato
            nodo_siguiente = nodo_siguiente.siguiente
        nodo_actual = nodo_actual.siguiente

    return self
```

Podemos notar que, a la hora de la implementación del método, tenemos dos bucles while anidados. El primero recorre cada nodo y el segundo va comparando los valores de los mismos. En caso de que el valor del nodo actual sea mayor al valor del nodo siguiente, se realiza un intercambio, y viceversa; ya que el ordenamiento de la lista se realizó de menor a mayor.
Para el mejor de los casos, en donde la lista esté ordenada, no se realizarán intercambios, pero para el peor, cada comparación de elementos causará un intercambio.
Como el método utiliza dos bucles anidados que se ejecutan n veces cada uno, lo que implica un tiempo de ejecución proporcional a n^2, podemos concluir que el orden de complejidad del algoritmo de ordenamiento implementado es **O(n2).**

Algoritmos y Estructuras de Datos

Gráfica del orden de complejidad de la función de ordenamiento