INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

MATERIA:

- Analisis y diseño de algoritmos

MAESTRA:

- Erika Sánchez Femat

ALUMNA:

- Naylin Yusseth Martínez Mireles

PRACTICA 02

(Implementación del método de Quicksort)

Índice

1. INTRODUCIÓN

La finalidad de hacer esta práctica, es poder entender e implementar el Quicksort, conocer como es que funciona de manera programable, ya que de manera manual ya lo pudimos resolver y hacer que quedara claro como era que su funcionamiento.
En el código lo que buscamos hacer es, lograr que ejecute 100 casos posibles, para de ellos poder comparar cuál es el mejor caso, el peor caso y el caso promedio, como lo hemos estado haciendo anteriormente.
De igual manera también poder interpretar cuál es la complejidad de este algoritmo nuevo y cuál es la diferencia ante otros que ya hemos programado.
El método de Quicksort es un algoritmo basado en la técnica, se divide y vencerás, que permite, en promedio, ordenar en elementos en un tiempo proporcional a n log n.
Este método fue creado por el científico británico Charles Antony Richard Hoare, también conocido como Tony Hoare en 1960, su algoritmo Quicksort es el algoritmo de ordenamiento más ampliamente utilizado en el mundo.
El método QuickSort es actualmente el más eficiente y veloz de los método de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición.
Sin más que abordar, mostraré como fue que lleve a cabo el desarrollo de este método.

2. DESARROLLO

En esta parte de la práctica, explicaré como es que implemente el código de Quicksort en el lenguaje que estamos manejando en la clase de análisis y diseño de algoritmos, este lenguaje es el ya conocido Python.
Lo primero que hice fue, comprender como es que funciona el algoritmo de manera manual. Sabemos que Quicksort utiliza la técnica, se divide y vencerás, esto para poder hacer más eficiente el código.

Entonces, supongamos que tenemos un arreglo de n números y de n longitud (esto de la longitud lo digo debido a que en la práctica, así lo solicita la maestra). Continuando con el arreglo, lo primero que vamos a hacer es saber cuál es la longitud de este arreglo, posterior, sumaremos los números que se encuentren dentro de él, para después dividir ese resultado entre la longitud del arreglo. En otras palabras, lo que haremos será calcular la media de dicho arreglo.

Una vez que obtengamos el resultado de esta operación, lo que haremos será tomar el valor resultante, como un "pivote"; en caso de que la operación arroje un resultado decimal o no exacto a los números que contiene el arreglo, lo que haremos será tomar el número que más se acerque.

Después de esto, usaremos ese "pivote"para seguir dividiendo el arreglo.

Lo que haremos será comparar todos los numero que se encuentren en el arreglo con el pivote obtenido. Aquí entra un parte importante del ordenamiento de este método, lo que tenemos que hacer una vez que comencemos a comparar, será mandar hacia la izquierda los números que sean menores al "pivotez de igual manera lo haremos con los números que sean mayores a este, a diferencia de que esos números los acomodaremos en la derecha.
Ahora vamos a repetir el primer paso para localizar los pivote en los nuevos arreglos que creamos.

Este algoritmo es recursivo, porque si recordamos la recursividad es una técnica que nos permite que un bloque de instrucciones se ejecute un cierto número de veces (el que nosotros determinemos).Entonces, debido a esto, si nos damos cuenta, notaremos que el algoritmo, hace lo mismo por un determinado número de veces, dependiendo de que tan desordenado esté

el arreglo.
Ahora, en manera programable, utilizando esto que ya explique, fue como implemente el código.

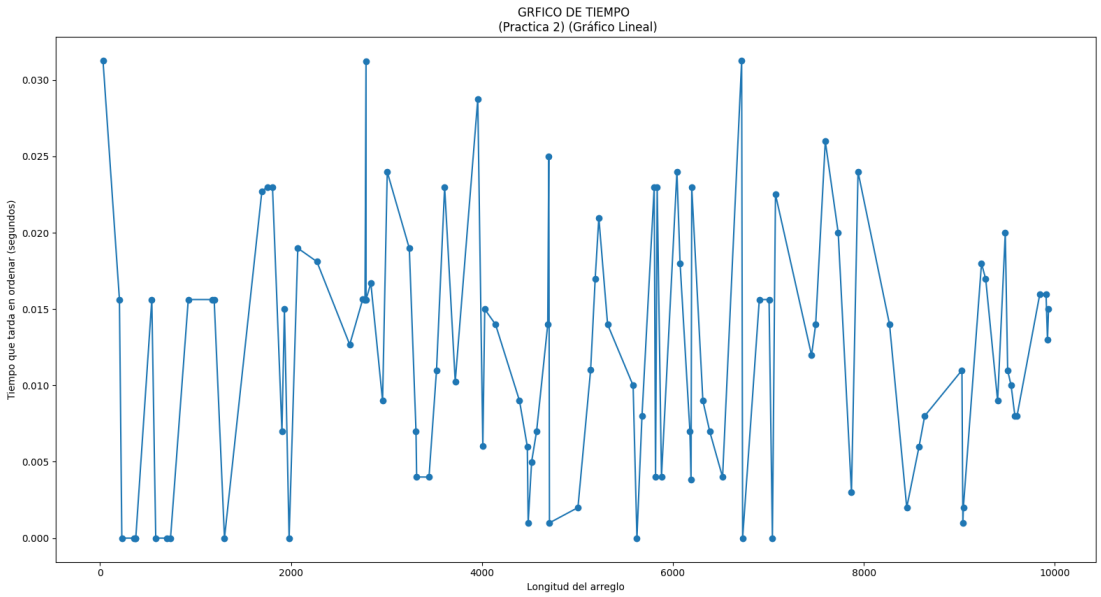
2.1. Analisis del código

```
#libreria del tiempo
import time
#libreria para generar numeros/arreglos aleatorios
import random
#libreria para graficar
import matplotlib.pyplot as plt

#definir una funcion para el ordenamiento QUICKSORT
def QUICKSORT(arr):

    #CASO BASE
    #cuando los arreglos sea igual a 1
    if len(arr) <= 1:
       #cuando esten ordenados ya los puede retornar
        return arr

     #CASO GENERAL
    #definir la suma de los elementos de arreglo
    suma = sum(arr)
    # Calcular la cantidad de elementos en el arreglo
    cantidad_elementos = len(arr)
     #para definir el pivote
    pivote = suma/cantidad_elementos
    #segunda manera de alegir el pivote
    #pivote = random.choice(arr)

    #Los elementos son menores, se van a la izquierda
    izquierda = [x for x in arr if x < pivote]

    #Los elemntos son mayores, se van a la derecha
```

```
    derecha = [x for x in arr if x > pivote]

    return QUICKSORT(izquierda) + QUICKSORT(derecha)

# hacemos una funcion para generar un arreglo aleatorio
def arreglo_aleatorio(n):
    return [random.randrange(10, 10000) for _ in range(n)]

# definimos la cantidad de arreglos que queremos
num_arreglos = 100
longitudes_arreglos = []
tiempos = []

for _ in range(num_arreglos):
    # Tamano aleatorio para el arreglo
    longitud_arreglo = random.randrange(10, 10000)
    arr = arreglo_aleatorio(longitud_arreglo)

    # Guardamos cuanto es que tarda en ordenar el arreglo
    inicio = time.time()
    tiempo = QUICKSORT(arr)
    final = time.time()

    longitudes_arreglos.append(longitud_arreglo)
    tiempos.append(final - inicio)

# Ordenar las longitudes de los arreglos
longitudes_arreglos.sort()

# Generar el grafico
plt.plot(longitudes_arreglos, tiempos, marker='o')
plt.xlabel("Longitud del arreglo")
plt.ylabel("Tiempo que tarda en ordenar (segundos)")
plt.title('GRAFICO DE TIEMPO \n (Practica 2) (Grafico Lineal)
plt.show()
```

Si observamos, en cada linea del código, tiene un comentario que explica como es que funciona y para que sirve cada renglon que esta escrito.

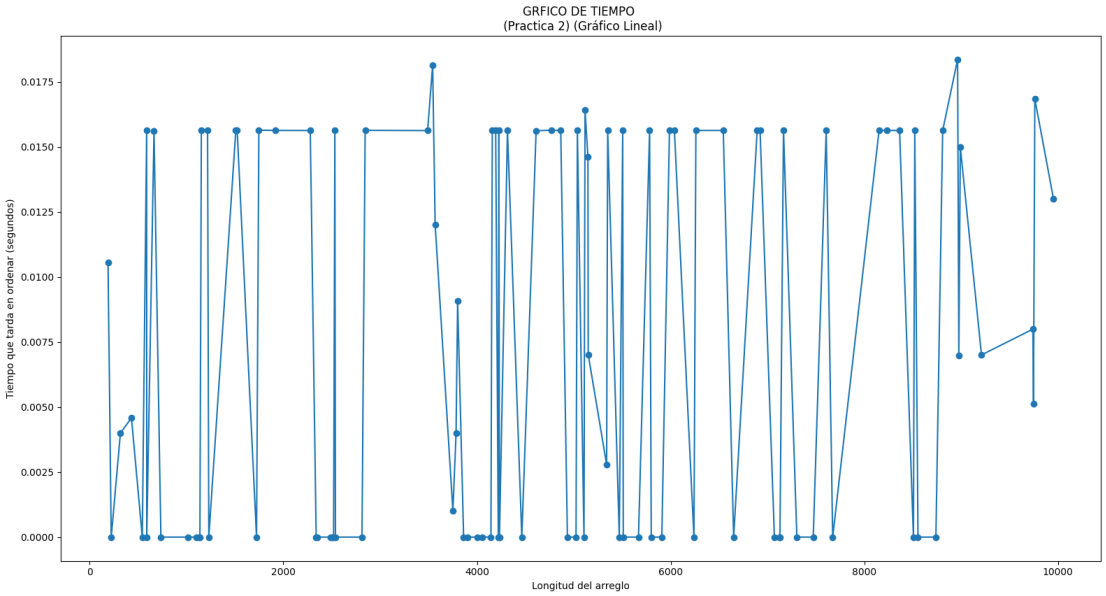
2.2. Analisis de la gráfica



En esta gráfica se expresa como es que todos los tipos de arreglos que se generan influye en el tiempo.
Cabe resaltar que esta gráfica es si tomamos como pivote lo que obtengamos de sacar la media.

3. MODIFICACIÓN DEL PIVOTE (2.5)

3.1. Analisis de la gráfica 2.0



En este caso, lo que hice en el codigo anterior, fue solo cambiar la manera de tomar el pivote, en el caso de la media, sumaba todos los elementos dentro del arreglo y los dividiamos entre la longitud de este arreglo.
Pero en este caso, tomamos como pivote, un elemento aleatorio, generado por el mismo código con la función, "$pivote = random.choice(arr)$".Que en el código original solo se encuentra comentado, para usar cualquiera de las dos opciones.

4. CONCLUSIÓN

En lo personal, este algoritmo, fue sumamente fácil de entender manualmente, a pesar de que en el código ingresamos una gran cantidad de datos, la compresión de este no se dificultó.
El método de ordenamiento de quicksort es normalmente más rápido que otros algoritmos de ordenación, como la ordenación por inserción o la ordenación por selección.

Además, es eficiente en términos de complejidad temporal y espacial.
Quicksort es también un algoritmo de ordenación muy flexible y puede adaptarse a diferentes estructuras de datos y tipos de datos.

En mi opinión podría decir que es muy interesante entender como es él funcionamiento de los distintos métodos de programar.
De igual manera, conocer más opciones de ordenamiento y así expandir el conocimiento sobre los lenguajes de programación estudiados.

Estos tipos de reportes o de información, son de gran utilidad, para ayudarnos a comprender como es el funcionamiento de las herramientas que tenemos como programadores y a su vez conocer cuáles de ellas nos pueden servir más y en que momento es necesario usarlas.

5. REFERENCIAS

http://mis-algoritmos.com/ordenamiento-rapido-quicksort

https://es.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort

https://www.neoteo.com/quicksort-algoritmo-de-ordenamiento-rapido/

https://tutospoo.jimdofree.com/tutoriales-java/m%C3%A9todos-de-ordeC3%B3n/ordenaci%C3%B3n-r%C3%A1pida-quicksort/

https://www.genbeta.com/desarrollo/implementando-el-algoritmo-quick

http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/mtodo_quick_sort.html