INSTITUTO POLITÉCNICO NACIONAL
Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

PRACTICA 2

MATERIA:
Ánalisis y Diseño de Algoritmos

DOCENTE:
Erika Sanchez Femat

GRUPO:
3CM2

ALUMNA:
America Lizbet Flores Alcala

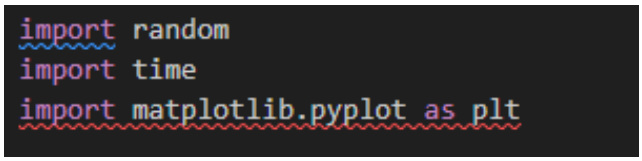
Introduccion

En la práctica siguiente, investigue, comprendi y aplique el algoritmo de QuickSort. Adquiri conocimiento sobre su funcionamiento y cómo implementarlo. Además, realicé modificaciones en el código siguiendo las pautas indicadas. Durante este proceso, explore nuevas habilidades, como la creación de gráficos en Python. Sin embargo, lo más destacado fue el logro de un funcionamiento correcto del código.

Desarrollo

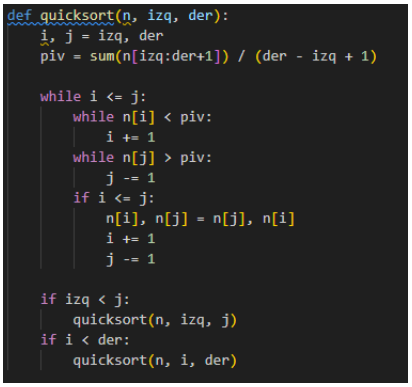
Ahora bien explicare un poco de lo que fue la implementacion del codigo QuikSort

1. Como primer paso estuvo importar las librerias que utilizamos, para generar numeros aleatorios(random), calcular el tiempo(time) y crear una grafica(matplotlib.pyplot)



```python
import random
import time
import matplotlib.pyplot as plt
```

2. Despues implemente el algoritmo de ordenacion, para esto defini el tamaño del arreglo *(n)* y dos indices que ayudarian a implementar este metodo *(izq, der)* y con estos datos se hace todo el proceso de ordenacion, aqui es importante destacar que como primera instancia el *pivote()* se eligio respecto de calcular la media del arreglo como se muestra en el codigo, como segunda instancia *(punto 2.5)* el *pivote()* se eligio respecto del numero que se encuentra a la mitad del arreglo, esto se logro despues de hacer algumas modificaciones al codigo principal

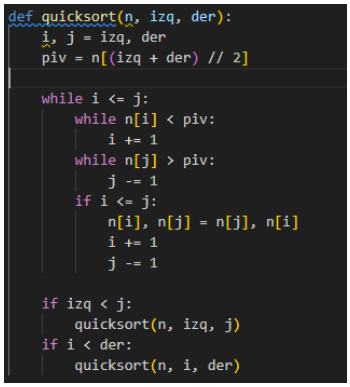


```python
def quicksort(n, izq, der):
    i, j = izq, der
    piv = sum(n[izq:der+1]) / (der - izq + 1)

    while i <= j:
        while n[i] < piv:
            i += 1
        while n[j] > piv:
            j -= 1
        if i <= j:
            n[i], n[j] = n[j], n[i]
            i += 1
            j -= 1

    if izq < j:
        quicksort(n, izq, j)
    if i < der:
        quicksort(n, i, der)
```

pivote sacando la media



```python
def quicksort(n, izq, der):
    i, j = izq, der
    piv = n[(izq + der) // 2]

    while i <= j:
        while n[i] < piv:
            i += 1
        while n[j] > piv:
            j -= 1
        if i <= j:
            n[i], n[j] = n[j], n[i]
            i += 1
            j -= 1

    if izq < j:
        quicksort(n, izq, j)
    if i < der:
        quicksort(n, i, der)
```

cogigo modificado respecto al punto 2.5

3. Como suiguiente paso cree dos listas vacías: tamaños_arreglos y tiempos_ejecucion. Estas listas se utilizarán para almacenar los tamaños de los arreglos y los tiempos de ejecución correspondientes, las cuales nos ayudaran al crear la grafica.

```
tamaños_arreglos = []
tiempos_ejecucion = []
```

4. Por consiguiente se crea un ciclo for, ya que como lo requeria la practica el metodo de ordenacion debe ejecutarse 100 veces, para esto se crea este ciclo, aqui se generan los numeros aleatorios del arreglo *(Lista Desordenada)* y tambien se implementa el metodo QuickSort para hacer la ordenacion de los mismos *(Lista Ordenada)* y como punto importante cabe destacar que el codigo calcula el tiempo que tarda en ordenarse cada arreglo *(Tiempo Ejecutado)*. En cada iteracion de este bucle el tamaño del arreglo y el tiempo de ejecución se agregan a las listas *tamaños_arreglos* y *tiempos_ejecucion* , respectivamente.

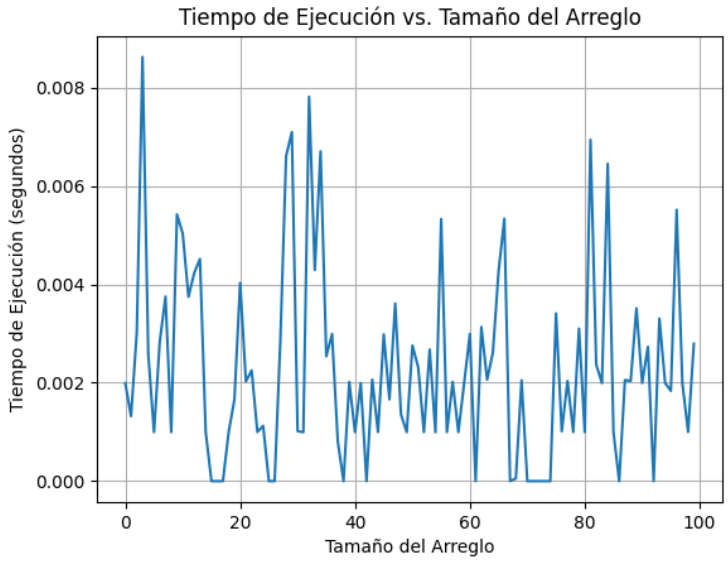
```
for i in range(0, 100):
    n = random.randint(10, 1000)
    aleatorios = [random.randint(0, 1000) for _ in range(n)]
    print("Lista Desordenada")
    print(aleatorios)

    inicio = time.time()
    quicksort(aleatorios, 0, len(aleatorios) - 1)
    fin = time.time()
    tiempo_ejecutado = fin - inicio
    print("Tiempo ejecutado", tiempo_ejecutado)

    print("Lista Ordenada")
    print(aleatorios)

    tamaños_arreglos.append(len(aleatorios))
    tiempos_ejecucion.append(tiempo_ejecutado)
```

5. Como ultimo paso, solo use funciones de la libreria que instale para crear la grafica que se muestra en la figura.



Explicacion de la grafica

La gráfica generada en el código anterior muestra cómo varía el tiempo de ejecución del algoritmo QuickSort en función del tamaño de los arreglos que se ordenan. El objetivo de este tipo de gráfica es evaluar el rendimiento del algoritmo en diferentes tamaños de entrada y observar cómo se comporta a medida que el tamaño del arreglo aumenta.
En el eje X, se representa el "Tamaño del Arreglo". Esto significa que se muestran los diferentes tamaños de los arreglos que se están ordenando. Cada punto en el eje X corresponde a un tamaño diferente de arreglo.
En el eje Y, se representa el "Tiempo de Ejecución" en segundos. Este eje muestra cuánto tiempo tardó el algoritmo QuickSort en ordenar un arreglo de un tamaño específico.

Conclusion

En conclusion, la experiencia de trabajar con el algoritmo Quicksort ha sido muy satisfactoria. Aprovechando la base previa que había adquirido en clases anteriores, pude entender sus fundamentos de manera más sólida. No obstante, la implementación práctica del código presentó algunos desafíos. La tarea de traducir mi comprensión teórica en un código funcional me expuso a desafíos comunes en programación, como errores de sintaxis, problemas con bibliotecas y dificultades de compilación. A pesar de estos obstáculos, logré implementar el algoritmo de manera eficiente. Esta experiencia me permitió no solo reforzar mis habilidades de programación, sino también ganar una apreciación más profunda de la eficiencia de Quicksort como un algoritmo de ordenación. A pesar de los desafíos, la sensación de logro al hacer que el algoritmo funcione correctamente es una bonita recompensa.

Referencias

https://parzibyte.me/blog/2020/09/08/quicksort-python-algoritmo-ordenamiento/

https://numython.github.io/posts/2016/02/graficar-en-python-con-matplotlib-y/