Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria de Ingeniería

Campus Zacatecas.

Práctica 2:

"Algoritmo de ordenamiento QuickSort".

Materia y Grupo:

Análisis y Diseño de Algoritmos. 3CM2.

Profesora a cargo:

Erika Paloma Sánchez-Femat.

Alumno:

Jorge Ulises Zapata Arteaga.

1 Introducción.

En este documento se estudiará la implementación del algoritmo de ordenamiento QuickSort con 100 arreglos aleatorios y elementos de los arreglos también aleatorios, con un recurso (gráfica) que nos será de utilidad para visualizar el tiempo utilizado para ordenar cada arreglo, así como los menores y mayores tiempos de ejecución.

2 Desarrollo.

- **Proceso de implementación del algoritmo.**

Entrada de ejemplo: [1, 3, 8, 16, 2, 4, 0, 1]
I) Debido a que la longitud del arreglo no es menor o igual a 1 elegimos el pivote de acuerdo a la media del arreglo, es decir (1+3+8+16+2+4+0+1)/8 = 4.3 y el elemento que más se acerca a ese número es 4, esto implica que el pivote es 4.
II) Ya elegido el pivote, inician las comparaciones para ver en qué arreglo de los 3 que se "crean" (*menor, igual o mayor*) se queda cada elemento restante del arreglo original, es decir, ¿1 >4?, no, entonces el 1 pasa al arreglo *menor*. ¿3 >4?, no, entonces 3 pasa al arreglo *menor*. ¿8 >4?, sí, entonces 8 pasa al arreglo *mayor*, siguiendo así con todos los demás elementos del arreglo, quedando:

[1, 3, 2, 0, 1] 4 [8, 16]

III) Ahora se escoge un pivote de cada arreglo *menor y mayor*. De parte del arreglo *menor* tenemos (1+3+2+0+1)/5 = 1.4 y el elemento que más se acerca a ese número es 1, esto implica que el pivote en *menor* es 1. Por otro lado, en el arreglo *mayor* tenemos (8+16)/2 = 12 y el elemento que más se acerca es 8, esto implica que el pivote en *mayor* es 8.
IV) Se sigue el mismo camino haciendo comparaciones de parte de ambos arreglos, quedando así:

[0, 1] 1 [3, 2] 4 8 [16]

V) Como último paso se elige un pivote para los arreglos menor y mayor del pivote 1, es decir (0+1)/2 = 0.5 y (3+2)/2 = 2.5, esto implica que el pivote en *menor* es 1 y el pivote en *mayor* es 2. Aplicando el mismo camino de comparaciones tenemos lo siguiente:

[0] 1 1 2 [3] 4 8 [16] = [0, 1, 1, 2, 3, 4, 8, 16]

Completando así el ordenamiento satisfactoriamente.

- **Proceso de implementación del código del algoritmo.**

Paso 1.- Para iniciar con la implementación del código del algoritmo, definimos una función *"quicksort"* y le asignamos un arreglo como entrada.
Paso 2.- Definimos el caso base que es cuando la longitud del arreglo es menor o igual a 1, esto nos retornará

el arreglo tal cual como se genera, si esto no se cumple definimos un pivote usando la media aritmética o promedio del arreglo, esto se logra sumando todos los elementos del arreglo y dividiendo esa suma entre la cantidad de elementos que hay.
Paso 3.- Creamos tres listas/arreglos, la primera llamada *menor*, la cual contendrá todos los elementos del arreglo original que son menores al pivote, para esto se recorre la lista/arreglo y seleccionamos los elementos que cumplen esta condición; la segunda llamada *igual*, para obtener los elementos del arreglo que son iguales al pivote, esto con la finalidad de trabajar los elementos duplicados que se encuentren en el arreglo; y por último la tercera llamada *mayor* que, al igual que las anteriores, guarda los elementos que son mayores al pivote.
Paso 4.- Para continuar definimos el caso general usando la misma función *quicksort* recursiva, concatenando los resultados de operación de los arreglos *menor, mayor e igual* para así obtener el arreglo ya ordenado.
Paso 5.- Definimos una nueva función llamada *arreglos* con un dato "tamano" como entrada, la cual nos sirve para generar arreglos desordenados con números enteros aleatorios con la librería "random".
Paso 6.- Declaramos una variable llamada *num arreglos=100* donde se define que vamos a generar 100 arreglos, además de la creación de un arreglo nuevo llamado *longitud Arreglos*, el cual se encarga de las longitudes de cada arreglo a generar. También declaramos dos arreg-

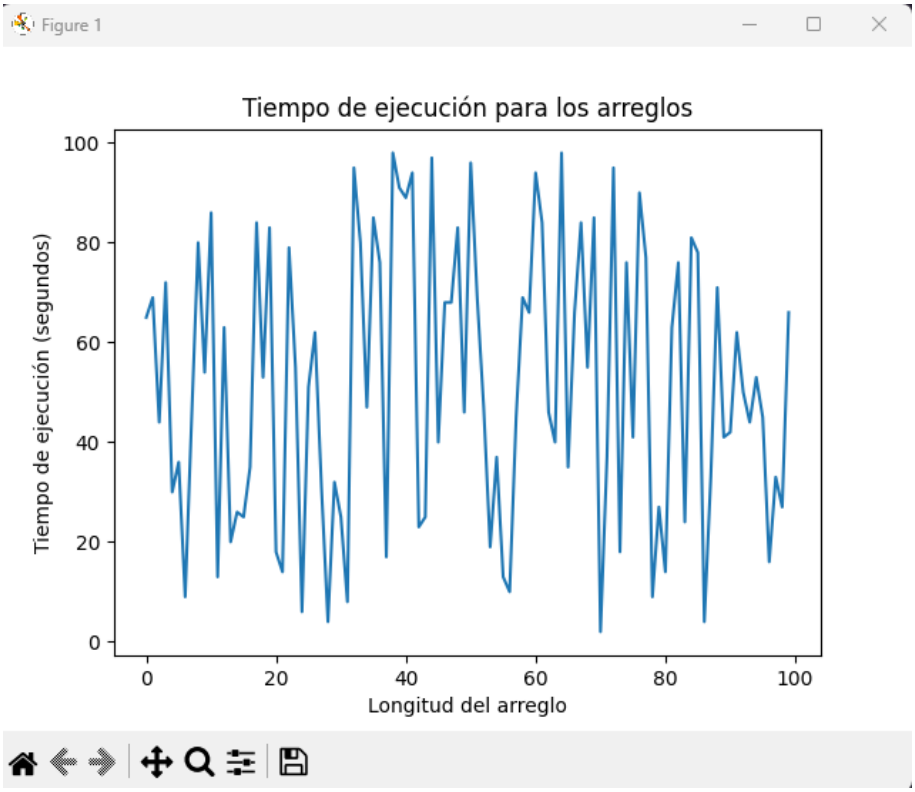
los vacíos (*x valores, y valores*), donde se almacenarán la longitud de los arreglos y el tiempo ejecutado, respectivamente.

Paso 7.- Imprimimos los arreglos aleatorios generados con un respectivo identificador para que, al momento de visualizarlo en consola, sea más fácil distinguir cada uno de ellos.

Paso 8.- Posteriormente medimos el tiempo (guardándolo en una variable *time*) que tarda el algoritmo en ordenar los 100 arreglos generados, para esto usamos la biblioteca "timeit", esta mide el tiempo de ejecución de la función quicksort, para esto hacemos uso de un argumento lambda que se le pasa a timeit y funciona para ejecutar por aparte la función quicksort 1000 veces (*number=1000*) midiendo el tiempo promedio. Por consiguiente, después de medir el tiempo, guardamos las longitudes de los arreglos en la lista *x valores* y los tiempos medidos en la lista *y valores*.

Paso 9.- Imprimimos los arreglos ya ordenados con su respectivo tiempo de ejecución usando la biblioteca numpy (*np.array()*), convirtiendo el arreglo ordenado por Quicksort en un objeto. Esto nos ayuda a imprimirlo de forma más legible.

Paso 10.- Mostramos la gráfica con los valores de (*x valores, y valores*) para ver más claramente los tiempos de ejecución dependiendo de cada longitud de los arreglos.



Gráfica de la salida del código.

En la gráfica anterior, podemos observar los tiempos de ejecución de cada longitud de arreglo.

3 Conclusiones.

Como hemos visto a lo largo de esta documentación, existe una manera muy eficiente de ordenar un arreglo o lista de elementos recursivamente, esta manera es el Algoritmo Quicksort, que se basa en la técnica de "divide y vencerás" por la que en cada recursión, el problema

se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos. Siendo este el algoritmo de ordenación más rápido de todos. En el reporte se muestran tanto la implementación del algoritmo (como funciona) como la implementación en código (programación) permitiéndonos entender y visualizar con mucha facilidad el proceso de ordenamiento.

4 Referencias.

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

https://www.youtube.com/watch?v=UrPJLhKF1jY