UNIDAD PROFESIONAL INTERDICIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS.

Práctica 2: QuichSort

Análisis y diseño de algoritmos

Autor:
Gloria Leticia González Troncoso.

22/Oct/2023

1 Introducción

Para esta práctica se busca conocer, comprender y saber implementar la forma de solución a un problema, basandose en la manera, ya vista, de la estrategia de "divide y conquista", así como ver la complejidad y qué tan eficiente es dentro de un algoritmo. Ahora el método ya investigado hace unos días, se ha logrado implementar ahora en código.

2 Desarrollo

En el desarrollo de este código se necesita tener en claro el cómo se implemeta este método, para ello se investigó y se hicieron pruebas simples para comprender de manera más sencilla este proceso.

Para empezar, al ser recursivo, es necesario definir su caso base, para esto el código empezará teniendo una función llamada "quicksort", donde mediante una condición se toma su caso base.

if len(arr) ¡= 1:
return arr

Dejando de lado el declarar arreglos, se declaran arreglos vacios, estos con el nombre de "mayor", "menor" e "igual". Estos servirar para almacenar los datos del arreglo de distintos lados ya eligiendo un "pivote", este sirve como clave para la organización de todo el arreglo. Dentro de esto se toma el pivote y los números menores a este se almacenan en menor, los mayores al pivote se almacenan dentro de mayor. Aqui este proceso se repite hasta que todo el arreglo quede organizado. A este proceso se toma como Caso general del código.

Para seguír se toman los arreglos y se mide el tiempo que le tomo a cada caso organizar. Esto con el fin de que dentro de una gráfica se tomen los datos.

Para poner o bien, que sea visible la gráfica se toma la libreria *import matplotlib.pyplot as plt*, y con las coordenadas tomadas de los datos del código, forman una gráfica que muestre los tiempos de ejecución que tiene cada arreglo para organizarse.

Su complejidad: En la práctica, es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O(n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable. El hecho de que sea más rápido que otros algoritmos de ordenación con tiempo promedio de O(n log (n)) (como SmoothSort o HeapSort) viene dado por que QuickSort realiza menos operaciones ya que el método utilizado es el de partición.

3 Conclusión

Para este análisis se demuestra que el método se puede implementar para varios casos en un mismo código, y su tiempo es poco y bastante rápido. Con ello se da cuenta de que los tiempos en los aque tarda cada caso no existe demaciada diferencia.

4 Referencias

https://manualdelatex.com/tutoriales/tipo-de-letra
https://www.udb.edu.sv/udb$_{f}iles/recursos_{g}uias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdfhttps://www.youtube.com/watch?v=UrPJLhKF1jYhttps://www.genbeta.com/desarrollo/implementando-el-algoritmo-quicksort$