Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas
Instituto Politécnico Nacional

Unidad 2 Estrategias de diseño de algoritmos

Práctica 02: Implementación y Evaluación del Algoritmo Quicksort

Alumno:
Kevin Marin Ramirez Reyna

Octubre 2023

1. Objetivo de la Práctica

El objetivo de esta práctica es el implementar el algoritmo Quicksort en Python, medir su rendimiento a través de la medición de tiempos de ejecución en arreglos aleatorios y visualicen los resultados mediante gráficos. Este enfoque permite comprender el funcionamiento del algoritmo, adquirir habilidades de programación, aprender a analizar datos y promover la experimentación y el pensamiento crítico en el contexto de la ordenación de datos.

2. Desarrollo de la Práctica

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.

2.1. Implementación Algoritmo Quicksort

Desarrollar el algoritmo de ordenamiento Quicksort en Python, teniendo en cuenta que la selección del pivote se debe hacer seleccionando la media del arreglo. Recuerda que este algoritmo es recursivo, por lo que tendrás que definir el caso base y el caso general.

La mayor parte de este algoritmo cae en la elección del pivote. Lo que se hace es tener dos puntos en los límites de la lista. El primero en el inicio, y el segundo en el final. Mientras la lista esté ordenada, vamos acercando estos puntos.

2.2. Generación de Casos de Prueba

Para comprender mejor el funcionamiento de este algoritmo, generamos 100 casos de prueba, es decir, generamos 100 arreglos de números aleatorios y de tamaño aleatorio. Con el fin de curbir la mayor cantidad de casos en los que este algoritmo se puede aplicar. Para este punto, generamos números aleatorios, con la librería random que ya viene incluida en la instalación de python.

2.3. Medicion del tiempo

Utilizamos parte que mida el tiempo de ejecución para cada caso de prueba.

Para ello iniciamos creando un arreglo para almacenar los tiempos, que se añadian despues de sacar el tiempo ejecutado que era restando el tiempo final - el inicial.

Creamos una lista para seguir los arreglos
arreglos = []

Iniciamos a medir el tiempo
inicio = time.time()
quicksort(arreglo)
fin = time.time()
tiempo ejecutado = fin - inicio
tiempos.append(tiempo ejecutado)

2.4. Visualizacion de Datos

import random
import time
import matplotlib.pyplot as plt

```
    Funcion para crear arreglos aleatorios
def arregloAleatorio():
Creamos una lista para guardar los tiempos
tiempos = []
Creamos una lista para guardar los tamaños
tamaños = []
Creamos una lista para seguir los arreglos
arreglos = []
for in range(0,100): Realizar 100 experimentos
Creamos Arreglos aleatorios
arreglo = [random.randint(0, 100) for i in range(random.randint(1, 100))]
tamaños.append(len(arreglo))
arreglos.append(arreglo)


    Iniciamos a medir el tiempo
inicio = time.time()
quicksort(arreglo)
fin = time.time()
tiempo ejecutado = fin - inicio
tiempos.append(tiempo ejecutado)

    Ordenamos los tamaños y tiempos en función de los tamaños
tamaños, tiempos, arreglos = zip(*sorted(zip(tamaños, tiempos, arreglos)))

    Graficamos después de haber recopilado los datos
plt.plot(tamaños, tiempos, marker='o', linestyle='-')
plt.title("Quicksort")
plt.xlabel("Tamaño del arreglo")
plt.ylabel("Tiempo en ejecución (segundos)")
plt.grid(True)
plt.show()

    Mostrar los 3 casos con menor tiempo
print(Çasos con menor tiempo de ejecución:")
for i in range(3):
print(f'Tamaño: tamaños[i], Tiempo: tiempos[i]')
print(f'Arreglo desordenado: arreglos[i]')
print(f'Arreglo ordenado: quicksort(arreglos[i])')

    Mostrar los 3 casos con mayor tiempo
print(Çasos con mayor tiempo de ejecución:")
for i in range(-1, -4, -1):
print(f'Tamaño: tamaños[i], Tiempo: tiempos[i]')
print(f'Arreglo desordenado: arreglos[i]')
print(f'Arreglo ordenado: quicksort(arreglos[i])')

    Quicksort
Funcion Quicksort
def quicksort(arreglo):
Cuando el arreglo solo tiene un valor estamos en el caso base
if len(arreglo)¡2:
Nos regresa la lista ordenada
return arreglo

    izq, pivote, der = particion(arreglo)
```

Nos ordena el lado izquierdo y derecho tambien
return quicksort(izq) + [pivote] + quicksort(der)

Funcion Auxiliar
def particion(arreglo):
Creamos un pivote que sea igualado a 0
pivote = arreglo[0]
Lista auxiliar para numeros menores
izq = []
Lista auxiliar para numeros mayores
der = []
Ciclo para separar los numeroes en mayores o menores, respecto al pivote
for i in range(1, len(arreglo)):
if arreglo[i]¡pivote:
izq.append(arreglo[i])
else :
der.append(arreglo[i])
return izq, pivote, der

print(quicksort(arreglo))

Metodo para ejecutar el programa
if name == 'main ' :
arregloAleatorio()

Figura 1: Quicksort

En esta grafica, podemos visualizar como es que fueron cada uno de los casos en el tama;o del arreglo, junto con el respectivo tiempo que tardaron en resolverse

Se toma como pivote el valor en la posicion 0, y utilizando la funcion auxiliar, lo vamos searando entre mayores y menores datos, utilizando recursividad volvemos a hacer en ambos lados tanto izquierda como derecha, para que al final solo nos quede el arreglo ordenado.

3. Conclusión

El metodo de quicksort, es el metodo mas rapido y eficientes de los metodos de ordenamiento, ya que con este es ampliamente utilizados para ordenar colecciones de elementos, con su tecnica de divide y veceras. con la cual va desglozando el arreglo tomando como base un pivote en el cual separa cuales sean mayores y menores que este, y recursivamente lo haga en ambos lados, para que asi este se ordene tambien en sus lados izquierdo y derecho,

4. Referencias

Implementando el algoritmo QuickSort (genbeta.com)

Quicksort en Python - Algoritmo de ordenamiento - Parzibyte's blog

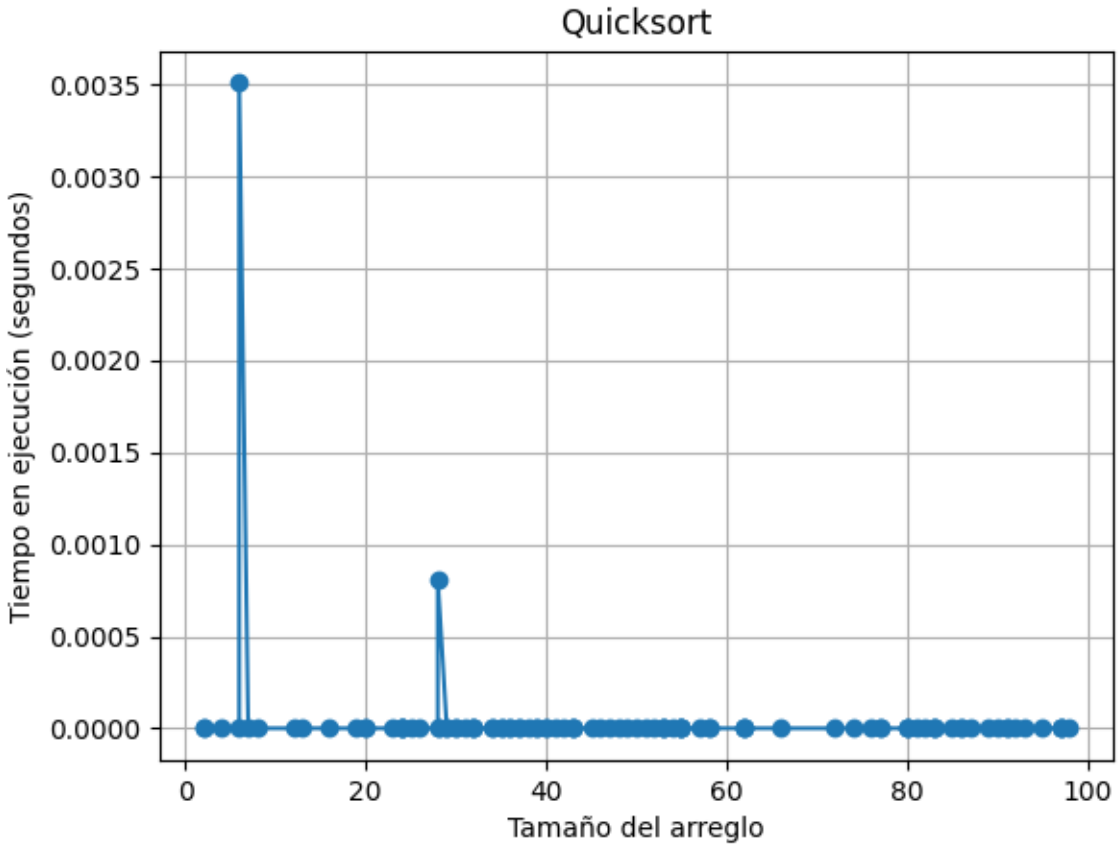

Figura 1: Quicksort