PRÁCTICA 2

ALGORITMOS DE ORDENAMIENTO II

## Alumno: Robles Martínez Héctor

## Número de Cuenta UNAM: 317356548

## Semestre: Tercer semestre de Ingeniería en Computación (2016)

## Semestre Actual: 2021-1

## Profesor: Jorge A. Solano G.

Objetivo:

Identificar la estructura de los algoritmos de ordenamiento Bubble Sort y Merge Sort.

Instrucciones:

• Implementar en Python los algoritmos de Bubble Sort y Merge Sort.

• Obtener los polinomios del mejor, el peor y el caso promedio de complejidad de cada algoritmo (Bubble sort y Merge sort).

• Graficar en Python el comportamiento de los algoritmos implementados para diferentes instancias de tiempo (listas de 1 a 1000 elementos) para el mejor (lista ordenada), el peor (lista ordenada en forma inversa) y el caso promedio (lista aleatoria) de complejidad.

Resultados obtenidos:

Bubble Sort:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 24 14:58:51 2020

@author: hrmha

"""

import random

def bubbleSort(data):

k = 0

n = len(data)

for i in range(n - 1):

i = i + 1

for j in range(n - 1):

i = i + 1

if data[j] > data[j + 1]:

data[j], data[j + 1] = data[j + 1], data[j]

k += 1

else:

k += 1

return data, k

def makeData(n):

lista = [0] \* n

for i in range(n):

lista[i] = random.randint(0, 1000)

return lista

def toPlotBs(n):

arr = makeData(n)

sortedData, complexNumP = bubbleSort(arr)

sortedDataB, complexNumB = bubbleSort(sortedData)

lstRvd = sortedDataB[::-1]

sortedDataW, complexNumW = bubbleSort(lstRvd)

return complexNumP, complexNumB, complexNumW

if \_\_name\_\_ == '\_\_main\_\_':

m, b, w = toPlotBs(10)

print("Middle: ", m, " Best: ", b, " Worse: ", w)

Merge Sort:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 24 14:58:51 2020

@author: hrmha

"""

import random

intI = 0

iteraciones = 0

def merge(left, right):

global iteraciones

iteraciones += 1

if not len(left) or not len(right):

return left or right

result = []

i, j = 0, 0

while (len(result) < len(left) + len(right)):

if left[i] < right[j]:

result.append(left[i])

i+= 1

else:

result.append(right[j])

j+= 1

if i == len(left) or j == len(right):

result.extend(left[i:] or right[j:])

break

return result

def mergesort(lista):

global intI, iteraciones

intI += 1

if len(lista) < 2:

return lista

middle = len(lista)//2

left= mergesort(lista[:middle])

right = mergesort(lista[middle:])

return merge(left, right)

def makeData(n):

lista = [0] \* n

for i in range(n):

lista[i] = random.randint(0, 100 \* n)

return lista

def toPlotMs(n):

global iteraciones, intI

lst = makeData(n)

mrg = mergesort(lst)

iteracionesN = iteraciones

intI = 0

iteraciones = 0

mrgOrd = mergesort(mrg)

iteracionesM = iteraciones

intI = 0

iteraciones = 0

lstRvd = mrgOrd[::-1]

intI = 0

mrgOrd = mergesort(lstRvd)

iteracionesP = iteraciones

return iteracionesN, iteracionesM, iteracionesP

if \_\_name\_\_ == "\_\_main\_\_":

m, b, w = toPlotMs(10)

print("Middle: ", m, " Best: ", b, " Worse: ", w)

Imagen que contiene texto, mapa

Descripción generada automáticamenteImagen de la gráfica de los tiempos en el orden:

|  |  |
| --- | --- |
| Promedio Merge Sort | Promedio Bubble Sort |
| Mejor Merge Sort | Mejor Bubble Sort |
| Peor Merge Sort | Peor Bubble Sort |

Conclusiones:

Esta práctica me sirvió para comprender cómo un algoritmo un poco más complicado de implementar, nos otorga mejores resultados que uno muy sencillo de implementar, siento esto un factor clave para decidir cuál de estos algoritmos usar.