Práctica 1

Estructura de datos y Algoritmos II

Algoritmos de Ordenamiento. Parte 1.

Objetivo: El estudiante identificará la estructura de los algoritmos de ordenamiento *Bubble Sort* y *Merge Sort*.

Antecedentes

- Análisis previo de los algoritmos en clase teórica.
- Manejo de arreglos o listas, estructuras de control y funciones en Python 3.

Desarrollo

Actividad 1

Actividad 1: Implementar en Python 3 los algoritmos de Bubble Sort y Bubble Sort Mejorado. La implementación debe mostrar el número de pasadas y los intercambios en cada pasada por ejemplo.

```
Algoritmo con mejora
pasada 1
[8, 20, 2, 39, 11, 34]
[8, 2, 20, 39, 11, 34]
[8, 2, 20, 39, 11, 34]
[8, 2, 20, 11, 39, 34]
[8, 2, 20, 11, 34, 39]
pasada 2
[2, 8, 20, 11, 34, 39]
[2, 8, 20, 11, 34, 39]
[2, 8, 11, 20, 34, 39]
[2, 8, 11, 20, 34, 39]
pasada 3
[2, 8, 11, 20, 34, 39]
[2, 8, 11, 20, 34, 39]
[2, 8, 11, 20, 34, 39]
[2, 8, 11, 20, 34, 39]
```

Los algoritmos son los siguientes:

```
bubbleSortSinMejora(A, n)
  Para i=1 hasta n -1
     Para j=0 hasta  n -2
       Si (a[j]>a[j+1]) entonces
        intercambia(a,j)
              Fin Si
     Fin Para
   Fin Para
Fin bubbleSort
```

```
bubbleSortMejorado (a, n)
    bandera = 1;
    pasada=0
    Mientras pasada < n-1 y bandera es  igual a 1
        bandera = 0
        Para j = 0 hasta  n-pasada-2
           Si  a[j] >a[j+1] entonces
               bandera = 1
               intercambia(A,j)
            Fin Si
         Fin Para
        pasada=pasada+1
    Fin Mientras
Fin
```

Probar ambos algoritmos con las secuencias 8,20,2,39,11,34,22,5,2,0 y 10,9,8,7,6,5,4,3,2,1

Abajo se muestra la implementación en Python de solo el Bubble sort sin mejora. Realizar el faltante.

```python
def bubbleSortSinMejora(A):
    n=len(A)
    for i in range(1,n):
        for j in range(0,n-1):
            if(A[j]>A[j+1]):
                intercambia(A,j,j+1)

def intercambia(A,j,i):
    tmp=A[j]
    A[j]=A[i]
    A[i]=tmp

def principal():
    A=[8,20,2,39,11,34,22,5,2,0]
    bubbleSortSinMejora(A)
    print(A)

principal()
```

De acuerdo a las salidas obtenidas responda lo siguiente

¿Qué se observa y se puede comentar al ordenar las dos secuencias con las dos funciones?

_____

¿Qué cambios se tienen que realizar al algoritmo para ordenar la lista en orden inverso? Describirlo y modificar el código. _____

Actividad 2 (Reporte): Las funciones proporcionadas para ordenar, ¿Realizan un burbujeo o hundimiento? Implementar el otro caso y mostrar cómo queda la secuencia en cada pasada

Actividad 3

A continuación, se proporciona el algoritmo Merge Sort, implementarlo en Python para ordenar una secuencia y agregar lo correspondiente para mostrar cómo se realiza la división de cada subsecuencia, así como las mezclas.

Ejemplo de salida para la secuencia 8,20,2,39,11,34

```
[8, 20, 2]
[8, 20]
[8]
[20]
Mezclar  izq [8]
Mezclar  der [20]
subsecuencia ordenada [8, 20]
[2]
Mezclar  izq [8, 20]
Mezclar  der [2]
subsecuencia ordenada [2, 8, 20]
[39, 11, 34]
[39, 11]
[39]
[11]
Mezclar  izq [39]
Mezclar  der [11]
subsecuencia ordenada [11, 39]
[34]
Mezclar  izq [11, 39]
Mezclar  der [34]
subsecuencia ordenada [11, 34, 39]
Mezclar  izq [2, 8, 20]
Mezclar  der [11, 34, 39]
subsecuencia ordenada [2, 8, 11, 20, 34, 39]
[2, 8, 11, 20, 34, 39]
```

Como ayuda se proporciona un código sugerido

```python
def MergeSort(A,p,r):
    if p<r:
        q=int((p+r)/2)
        MergeSort(A,p,q)
        MergeSort(A,q+1,r)
        Mezcla(A,p,q,r)

def Mezcla(A,p,q,r):
    izq=A[p:(q+1)]
    der=A[q+1:r+1]
    i=j=0
    for k in range(p,r+1):
        if (j>=len(der)) or (i< len(izq) and izq[i]< der[j]):
            A[k]=izq[i]
            i+=1
        else:
            A[k]=der[j]
            j+=1
```

Actividad 4: Realiza un programa que ordene utilizando los pseudocódigos de insertion sort y selection sort. Además, se muestre su funcionamiento (que sucede en cada pasada)

```
ordenacionPorSeleccion(A)
Inicio
 Desde i=0 hasta  n-2 hacer
   min=i;
        Desde j=i+1 hasta n-1 hacer
                Si  A[j] < A[min] entonces
                        min=j
                fin si
        Fin Desde
        Intercambia(A,i,min)
  FinDesde
Fin ordenacionPor Seleccion
```

```
InsertionSort(A,n)
   Para j=1 hasta n-1
      llave=A[j]
                i=j-1
                Mientras i >= 0 y A[i] > llave
                        A[i+1]=A[i]
                        i=i-1
      Fin Mientras
            A[i+1]=llave
   Fin Para
Fin
```

Ejemplo Salida Insertion Sort

```
pasada 1 elemento a insertar 9
[12, 9, 3, 7, 14, 11]
[12, 12, 3, 7, 14, 11]
[9, 12, 3, 7, 14, 11]
pasada 2 elemento a insertar 3
[9, 12, 3, 7, 14, 11]
[9, 12, 12, 7, 14, 11]
[9, 9, 12, 7, 14, 11]
[3, 9, 12, 7, 14, 11]
pasada 3 elemento a insertar 7
[3, 9, 12, 7, 14, 11]
[3, 9, 12, 12, 14, 11]
[3, 9, 9, 12, 14, 11]
[3, 7, 9, 12, 14, 11]
pasada 4 elemento a insertar 14
[3, 7, 9, 12, 14, 11]
[3, 7, 9, 12, 14, 11]
pasada 5 elemento a insertar 11
[3, 7, 9, 12, 14, 11]
[3, 7, 9, 12, 14, 14]
[3, 7, 9, 12, 12, 14]
[3, 7, 9, 11, 12, 14]
[3, 7, 9, 11, 12, 14]
```

Ejemplo Salida Selection Sort

```
Lista original [12, 9, 3, 7, 14, 11]
Pasada 0
[12, 9, 3, 7, 14, 11]
    Elemento mínimo seleccionado 3
[3, 9, 12, 7, 14, 11]
Pasada 1
[3, 9, 12, 7, 14, 11]
    Elemento mínimo seleccionado 7
[3, 7, 12, 9, 14, 11]
Pasada 2
[3, 7, 12, 9, 14, 11]
    Elemento mínimo seleccionado 9
[3, 7, 9, 12, 14, 11]
Pasada 3
[3, 7, 9, 12, 14, 11]
    Elemento mínimo seleccionado 11
[3, 7, 9, 11, 14, 12]
Pasada 4
[3, 7, 9, 11, 14, 12]
    Elemento mínimo seleccionado 12
[3, 7, 9, 11, 12, 14]
Pasada 5
[3, 7, 9, 11, 12, 14]
    Elemento mínimo seleccionado 14
[3, 7, 9, 11, 12, 14]
[3, 7, 9, 11, 12, 14]
```

Actividad 5 (Reporte). Realizar un programa que ordene por Merge Sort utilizando lo salgoritos en pseudocodigo que se muestran. Es importante analizar y entender el algoritmo para que se pueda implementar.

```
mergesort(s, n):
    if n ≤ 1 then:
        return s
    end if
    mid := ⌊n/2⌋
    s1 := arreglo de tamaño mid+1
    s2 := arreglo de tamaño n-mid+1
    copia s[0]...s[mid-1] a s1
    copia s[mid]...s[n-1] a s2
    mergesort(s1, mid+1)
    mergesort(s2, n-mid+1)
    s1[mid] := ∞
    s2[n-mid] := ∞
    merge(s1, s2, s, n)
    return s
```

Algoritmo Merge Sort

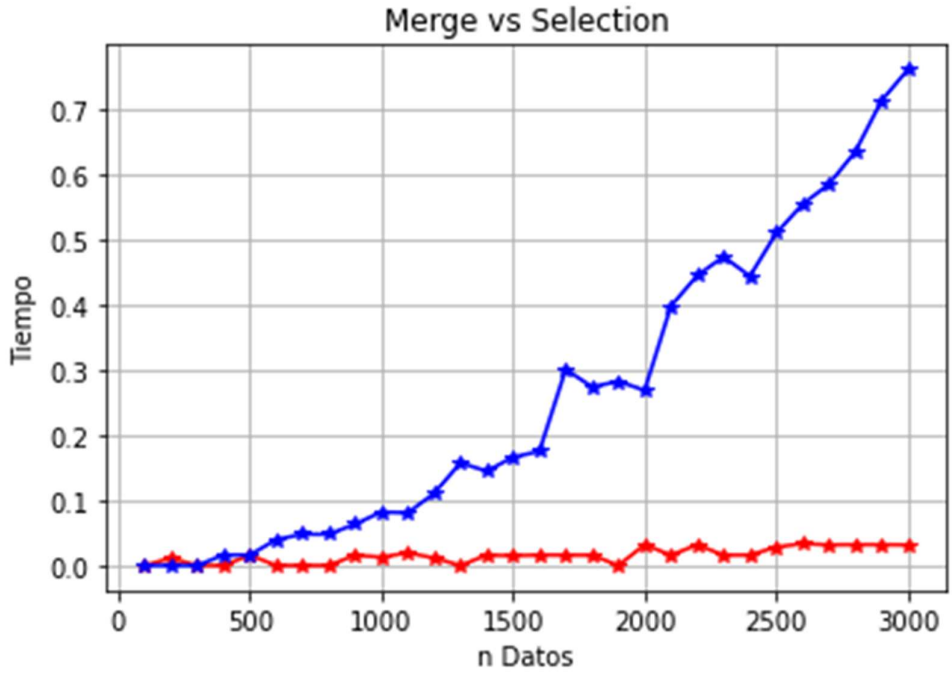
```
merge(s1, s2, s, n):
    i := 0
    j := 0
    while i+j < n:
        if s1[i] < s2[j] then:
            s[i+j] := s1[i]
            i := i + 1
        else:
            s[i+j] := s2[j]
            j := j + 1
        end if
    end while
```

Actividad 6

Se proporciona un ejemplo donde se grafica tiempos de ejecución de la ordenación de listas generadas de tamaños de 100 a 3000 elementos (de 100 en 100) para el algoritmo de bubble sort sin mejora

Se pide agregar lo necesario para graficar el ordenamiento de las mismas listas pero ordenadas con BubbleSortMejorado, selection sort , insertion sort.

Un ejemplo es el siguiente, pero con todas las gráficas poder realizar la comparación.




```python
import matplotlib.pyplot as plt
import random
from time import time
def graficas():
    TM=[]
    nD = [k*100 for k in range (1,31)]

    for k in nD:
        datosM=random.sample(range(1,100000000),k)
        t0=time()
        MergeSort(datosM,0,len(datosM)-1)
        t1=time()
        TM.append(round(t1-t0,6))


    fig,ax=plt.subplots()

    ax.plot(nD, TM,label="Merge Sort", marker= "*", color="r")
    ax.set_xlabel("n Datos")
    ax.set_ylabel("Tiempo")
    ax.grid(True)
    plt.title("Merge  Sort")
    plt.show()

graficas()
```