Profesora: Elba Karen Sáenz

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*.

Al final de la práctica el estudiante habrá implementado los algoritmos en algún lenguaje de programación.

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

Abajo se muestra la implementación en Python de los pseudocódigos de las funciones $bubbleSort()$ y $bubbleSort2()$ vistas en clase que permiten realizar el ordenamiento de una lista por ***BubbleSort***. Se pide realizar un programa que ordene una lista de n elementos utilizando ambas funciones. Se debe probar con las siguientes secuencias.

1- 8,20,2,39,11,34
2- 10,9,8,7,6,5,4,3,2,1

```python
def bubbleSort(A):
    for i in range(1,len(A)+1):
        for j in range(len(A)-1):
            if A[j]>A[j+1]:
                temp = A[j]
                A[j] = A[j+1]
                A[j+1] = temp


def bubbleSort2(A):
    bandera= True
    pasada=0
    while pasada < len(A)-1 and bandera:
        bandera=False
        for j in range(len(A)-1-pasada):
            if(A[j] > A[j+1] ):
                bandera=True
                temp = A[j]
                A[j] = A[j+1]
                A[j+1] = temp
        pasada = pasada+1
```

Actividad 2: Agregar al código la impresión para conocer el número de pasadas y los intercambios en cada pasada para cada función

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]
```

¿Qué diferencia se encuentran entre las dos funciones al ordenar las listas con las dos funciones y listas de la actividad anterior?

_____

Qué se tiene que hacer para ordenar la lista en orden inverso? Describirlo y modificar el código.

_____

Actividad 3: Las funciones proporcionadas para ordenar, ¿Realizan un burbujeo de los elementos más pequeño a la parte superior(inicio) de la lista o un hundimiento de los elemento mayores a la parte inferior(fin)?. Indicar_____

Modificar el programa para realizar el otro caso dependiendo de la respuesta.

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-1 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
                Mientras i > 0 y A[i-1] > llave
                      A[i]=A[i-1]
                      i=i-1
       Fin Mientras
            A[i]=llave
   Fin Para
Fin
```

Actividad 5 (para reporte)

Realizar un programa que ordene por *Shaker sort*

Actividad 5

A continuación, se proporciona la implementación en Python de los pseudocódigos de las funciones dadas en clase para el algoritmo ***MergeSort***. Se requiere utilizarlas para elaborar un programa que ordene una lista de n elementos.

Agregar en el lugar correspondiente la impresión, para visualizar las sub-secuencias obtenidas en la recursión antes de mezclar y después las obtenidas al ir mezclando.

```python
#MergeSort

def CrearSubArreglo(A, indIzq, indDer):
    return A[indIzq:indDer+1]

def Merge(A,p,q,r):
    Izq = CrearSubArreglo(A,p,q)
    Der = CrearSubArreglo(A,q+1,r)
    i = 0
    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 = i + 1
        else:
            A[k] = Der[j]
            j = j + 1

def MergeSort(A,p,r):
    if r - p > 0:
        q = int((p+r)/2)
        MergeSort(A,p,q)
        MergeSort(A,q+1,r)
        Merge(A,p,q,r)
```

Actividad 6

¿Qué cambio(s) se hace(n) en el algoritmo para ordenar la lista en orden descendente? . Describirlo y realizar cambios en el código.

Actividad 7 Realizar un programa que ordene por Merge Sort utilizando los algoritmos en pseudocódigo que se muestran a continuación.

```
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 8 (Para Reporte)

Se proporciona una función donde se grafica en Anaconda tiempos de ejecución de la ordenación de listas generadas de tamaños de 100 a 3000 elementos (de 100 en 100), utilizando dos algoritmos de ordenamiento Bubble Sort y Selection Sort

Se pide graficar el ordenamiento de las mismas listas para BubbleSort2,Selection Sort, Insertion y MergeSort

```
#############################################################################################
%pylab inline
import matplotlib.pyplot as plt
import random
from time import time
def grafica():
    TiempoS = []
    Tiempo = []
    nDatos = [k*100 for k in range (1,31)]

    for k in nDatos:
        datosOr = random.sample(range(0,10000000),k)
        datosSS = datosOr
        t0 = time()
        MergeS(datosOr, 0, len(datosOr)-1)
        Tiempo.append(round(time()-t0,6))
        #
        t0 = time()
        OrdenPorSeleccion(datosSS)
        TiempoS.append(round(time()-t0,6))

    #GRAFICA MERGE
    fig,ax = subplots()
    ax.plot(nDatos, Tiempo, label = "MergeS", marker = "*", color = "r")
    ax.set_xlabel('Datos')
    ax.set_ylabel('Tiempo')
    ax.grid(True)
    plt.title('Merge: Datos vs Tiempo');
    plt.show()

    #GRAFICA ORDENACION seleccion
    fig,ax = subplots()
    ax.plot(nDatos, TiempoS, label = "OrdenPorSeleccion", marker = "*", color = "b")
    ax.set_xlabel('Datos')
    ax.set_ylabel('Tiempo')
    ax.grid(True)
    plt.title('OrdenPorSeleccion: Datos vs Tiempo');
    plt.show()

    #GRAFICAS JUNTAS
    fig,ax = subplots()
    ax.plot(nDatos, Tiempo, label = "MergeS", marker = "*", color = "r")
    ax.plot(nDatos, TiempoS, label = "OrdenPorSeleccion", marker = "*", color = "b")
    ax.set_xlabel('Datos')
    ax.set_ylabel('Tiempo')
    ax.grid(True)
    plt.title('MergeS(ROJO) VS Ordenacion(AZUL)');
    plt.show()
grafica()
```

Conclusiones