	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	11/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Desarrollo

Actividad 1

Abajo se muestra la implementación en Python de los pseudocódigos de las funciones *bubbleSort()* y *bubbleSort2()* mencionados en la guía y que permiten realizar el ordenamiento de una lista por **BubbleSort**. Se pide realizar un programa que ordene una lista de n elementos (la cual es proporcionada por el profesor) utilizando ambas funciones.


```
#Función BubbleSort
#Autor Elba Karen Sáenz García

def bubbleSort(A):
    for i in range(1, len(A)):
        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

#Función BubbleSort Mejorada
#Autor Elba Karen Sáenz García
def bubbleSort2(A):
    bandera= True
    pasada=0
    while pasada < len(A)-1 and bandera:
        bandera=False
        for j in range(len(A)-1):
            if(A[j] > A[j+1]):
                bandera=True
                temp = A[j]
                A[j] = A[j+1]
                A[j+1] = temp
        pasada = pasada+1
```

Agregar al código la impresión para conocer el número de pasadas que realiza cada función e indicarlo. _____

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

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	12/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Actividad 2

A continuación, se proporciona la implementación en Python de los pseudocódigos de las funciones mencionadas en la guía para el algoritmo **MergeSort**. Se requiere utilizarlas para elaborar un programa que ordene una lista proporcionada por el profesor.

Agregar en el lugar correspondiente la impresión, para visualizar las sub-secuencias obtenidas en la recursión.

```
#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)
```

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

Actividad 3

Ejercicios propuestos por el profesor.