```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 26 16:42:44 2021

@author: itzamadg
"""
"""
Paquetes
"""

#Para manejo de vectores
import numpy as np
#Para graficar
from matplotlib import pyplot as plt
#Para medir tiempo de ejecución
from timeit import default_timer as timer
#Para la generación de arreglos de strings
#aleatorios
import string
import random
"""
-----------------------------------------------------------
A continuación los algoritmos de ordenamiento
"""
#Este número será la base para el tamaño de los arreglos
#3^1,3^2,... etc
base = 3


def selectionSort(A):
    """


    Parameters
    ----------
    A : Un arreglo
    de elementos con
    una relación de orden

    Returns
    -------
    comparacion : Número
    de comparaciones que realiza el método

    """
    comparacion = 0
    for i in range(len(A)-1):
        min_pos=i
        for j in range(i+1,len(A)):
            comparacion+=1
            if(A[min_pos]>A[j]):

                min_pos = j

        temporal = A[i]
        A[i] = A[min_pos]
        A[min_pos] = temporal
```

```
    return comparacion

#Mismos parámetros
def insertionSort(A):
    comparacion = 0
    for i in range(1,len(A)):
        j = i
        comparacion+=1
        while(j>=1 and A[j]<A[j-1]):
            comparacion+=1
            temp = A[j]
            A[j]=A[j-1]
            A[j-1]=temp
            j-=1
    return comparacion

#Metodo recursivo
def mergeSortR(A,m,M):
    comparacion = 0
    if(M-m>=1):
        comparacion = 0
        mid = (M+m)//2
        comparacion = mergeSortR(A,m,mid)
        comparacion = comparacion + mergeSortR(A,mid+1,M)

        B = [0]*(M-m+1)
        i = m
        j = mid+1
        k = 0


        while(i<=mid and j<=M):
            comparacion+=1
            if(A[i]<=A[j]):
                B[k] = A[i]
                i+=1
            else:
                B[k] = A[j]
                j+=1
            k+=1

        while(i<=mid):
            B[k] = A[i]
            i+=1
            k+=1

        while(j<=M):
            B[k] = A[j]
            j+=1
            k+=1

        for p in range(len(B)):
            A[m+p]=B[p]
    return comparacion

#Metodo que llama al recursivo
```

```
#Mismos parámetros que los anteriores
def mergeSort(A):
    return mergeSortR(A,0,len(A)-1)
#Mismos parámetros
def bubbleSort(A):
    sorted_a  = False
    comparacion = 0
    while(not(sorted_a)):
        sorted_a = True

        for i in range(len(A)-1):
            comparacion+=1
            if(A[i]>A[i+1]):
                temporal = A[i]
                A[i] = A[i+1]
                A[i+1] = temporal
                sorted_a = False
    return comparacion

#Recibe un arreglo y dos
#posiciones, intercambia
#los elementos de estas posiciones
def swap(A,i,j):

    temp = A[i]
    A[i]=A[j]
    A[j] = temp



def particion(arreglo, m, M):
    """


    Parameters
    ----------
    arreglo : Un arreglo
    de elementos con una
    relación de orden
    m : Extremo izquierdo
    del arreglo
    M : Extremo derecho del
    arreglo

    Returns
    -------
    i : Posición
    final del pivote.

    """
    i = m
    j = M
    while(i<j):
        if(arreglo[i+1]<=arreglo[i]):
            swap(arreglo,i,i+1)
            i+=1
        else:
```

```
            swap(arreglo,i+1,j)
            j-=1

    return i

#Método recursivo
def quickSortR(arreglo, m, M):
    comparacion = 0
    if(M-m<=1):
        return comparacion

    pivote = particion(arreglo,m,M)
    comparacion +=1 #La correspondiente a lo dentro de particion
    comparacion = comparacion + quickSortR(arreglo,m,pivote-1)
    comparacion = comparacion+ quickSortR(arreglo,pivote+1,M)
    return comparacion
    # quickSortR(arreglo,m,pivote-1)
    # quickSortR(arreglo,pivote+1,M)
#Método que manda a llamar al recursivo
#Mismos parámetros
def quickSort(A):
    return quickSortR(A,0,len(A)-1)
    # quickSortR(A,0,len(A)-1)

"""
-------------------------------------------------------------------------------
"""
#Establecemos la semilla con la
#que se generarán los números
#pseudoaleatorios
np.random.seed(111013)


def copia(Samples):
    """
    Parameters
    ----------
    Samples : Lista
        de arreglos a ordenar

        Usaremos esta funcion para que todos
        puedan ordenar los mismo arreglo


    Returns:

    -------
    Regresa 5 copias de Samples, una para cada método
    Es decir, cinco listas con arreglos de distintos
    tamaños

    """
    samples_selection =[]
    samples_insertion =[]
    samples_quick = []
    samples_merge = []
    samples_bubble =[]
```

```
    for i in range(len(Samples)):
        samples_selection.append(Samples[i].copy())
        samples_insertion.append(Samples[i].copy())
        samples_quick.append(Samples[i].copy())
        samples_merge.append(Samples[i].copy())
        samples_bubble.append(Samples[i].copy())
    result = [samples_selection,samples_insertion,samples_quick,samples_merge,
              samples_bubble]

    return result
def Ordena(M,Samples):

    """
    Parameters
    ----------
    M : Método
    de ordenamiento

   Samples : Lista
        de arreglos a ordenar

    Returns
    A vector with size of each sample
    A vector with the time took to process each sample
    -------
    None.

    """
    #Aqui guardaremos el tamaño del arreglo a ordenar
    size =[]

    #Aqui guardaremos el tiempo para ese arreglo
    time = []

    #comparaciones
    comp = []
    for sample in Samples:
        size.append(len(sample))
        start = timer()
        comp.append(M(sample))
        end = timer()
        time.append(end-start)

    return size,time,comp

def randomSamples(n,indicador = 0):



    """
    Parameters
    ----------
    n : Potencia máxima base^n

    indicador: 0 para trabajar
    floats, 1 para trabajar integers
```

```
    Returns
    -------
    Una lista de n arreglos distintos
    generados aleatoriamente, donde el
    i-ésimo arreglo tiene tamaño base^i
    Para i= 1,2,3,...,n

    """
    random_samples = []
    if(indicador == 0):
        for i in range(1,n+1):
            random_samples.append(1+10*np.random.random_sample(base**i))
    elif(indicador == 1):
        for i in range(1,n+1):
            random_samples.append(np.random.randint(-base**i,base**i,base**i))

    return random_samples



def orderedSamples(n,indicador=0):
    """


    Parameters
    ----------

    n :

    indicador:

    n : Potencia máxima base^n
    indicador :  0 para trabajar
    floats, 1 para trabajar integers

    Returns
    -------
    ordered_samples : Una
    lista de arreglos ordenados


    """


    ordered_samples = []
    if(indicador == 0):
        for i in range(1,n+1):
            #Primero creamos la muestra aleatoria
            temporal = 1+10*np.random.random_sample(base**i)
            #la ordenamos
            temporal.sort()
            #la guardamos
            ordered_samples.append(temporal)

    elif(indicador == 1):
        for i in range(1,n+1):
            ordered_samples.append(np.arange(base**i))
```

```
    else:
        ordered_samples = randomStringSample(n)
        for sample in ordered_samples:
            sample.sort()



    return ordered_samples


def inversedSamples(orderedSamples):
    """
    Parameters
    ----------
    orderedSamples : Requiere
        una lista de arreglos, que estos
        arreglos esten ordenados, para regresar una
        lista con los mismos arreglos pero ordenados
        de forma inversa

    Returns
    -------
    Una lista de arreglos con orden inverso

    """
    reverseSamples = []
    for sample in orderedSamples:
        reverseSamples.append(sample[-1:-(len(sample)+1):-1])

    return reverseSamples

def randomString(n):
    """


    Parameters
    ----------
    n : tamaño del
    string aleatorio

    Returns
    -------
    st : Un string
    aleatorio de tamaño n
    """
    st=''.join(random.choice(
        string.ascii_uppercase + string.digits) for _ in range(n))
    return st

def randomStringSample(n):
    """


    Parameters
    ----------
    n : Potencia para el tamaño
```

```
    máximo del arreglo base^n.

    Returns
    -------
    Una lista de arreglos de Strings
    aleatorios de tamaño base^i para
    i=1,...,n. Los strings tienen
    tamaño entre 1 y 10

    """
    #aqui guardaremos los n arreglos
    random_String = []
    for i in range(1,n+1):
        #El temporal para el arreglo de tamaño i
        sample = []

        #Para el i-ésimo arreglo
        #Hay que añadir base**i strings
        for j in range(base**i):
            sample.append(randomString(np.random.randint(1,10)))
        random_String.append(sample)
    return random_String

def graficaSolo(M,indicador,n):
    """
    Parameters
    ----------
    M : Es el método a
    usar para el ordenamiento,
    y posteriormente graficar

    indicador : 0 para ordenar
    arreglos en orden inverso, 1 para
    ordenar arreglos en orden aleatorio
    y 2 para ordenar arreglos ordenados

    n : Potencia del tamaño máximo
    del arreglo, se ordenan arreglos
    de hasta base^n elementos

    Returns
    -------
    No regresa nada, grafica, dependiendo del indicador
    para arreglos de 3 tipos: Floats, Integers, Strings

    """

    if(indicador!=0 and indicador !=1 and indicador != 2):
        return

    if(indicador == 0):
        #Orden inverso

        revSamp = inversedSamples(orderedSamples(n,0))
        revSamp1 = inversedSamples(orderedSamples(n,1))
        revSamp2 = inversedSamples(orderedSamples(n,2))
```

```
    size,time,comp = Ordena(M,revSamp)
    size1,time1,comp1 = Ordena(M,revSamp1)
    size2,time2,comp2 = Ordena(M,revSamp2)
    title = "Orden inverso: "



elif(indicador == 1):
    #Orden aleatorio
    size,time,comp = Ordena(M,randomSamples(n,0))
    size1,time1,comp1 = Ordena(M,randomSamples(n,1))
    size2,time2,comp2 = Ordena(M,randomStringSample(n))
    title = "Orden Aleatorio: "
elif(indicador ==2):
    size,time,comp = Ordena(M,orderedSamples(n,0))
    size1,time1,comp1 = Ordena(M,orderedSamples(n,1))
    size2,time2,comp2 = Ordena(M,orderedSamples(n,2))
    title = "Datos Ordenados: "


if(M==insertionSort):
    title+= "Insertion Sort"
elif(M==selectionSort):
    title+= "Selection Sort"

elif(M == quickSort):
    title+= "Quick Sort"
elif(M == bubbleSort):
    title+="Bubble Sort"
elif(M == mergeSort):
    title+= "Merge Sort"

legend = ["Floats","Integers","Strings"]
figure = plt.figure()
figure.suptitle(title, fontsize=16)


#Grafico izquierdo
# ax = plt.subplot(1,2,1)
# ax.plot(size,comp)
# ax.plot(size1,comp1)
# ax.plot(size2,comp2)
# ax.set_xlabel("Número de elementos")
# ax.set_ylabel("Comparaciones")
# ax.grid()
# ax.legend(legend)

ax1 = plt.subplot(1,1,1)
ax1.plot(size,time)
ax1.plot(size1,time1)
ax1.plot(size2,time2)
ax1.set_xlabel("Número de elementos")
ax1.set_ylabel("Tiempo-Segundos")
ax1.grid()
ax1.legend(legend)
```

```
def graficaConjunto(indicador,n,floats=0):
    """


    Parameters
    ----------
    indicador : Análogo a la función
    anterior
    n : Análogo a la función anterior
    floats : Análogo a la función anterior

    Returns
    -------
    Realiza dos gráficas, una grafica
    #n_elementosvsnumero de operaciones y
    #n_elementos vs tiempo de ejecución
    de todos los
    métodos, según el indicador que se ponga.

    """
    if(indicador!=0 and indicador !=1 and indicador != 2):
        return

    Metodos = [selectionSort,insertionSort,quickSort,mergeSort,bubbleSort]
    Size = []
    Time = []
    Comp = []
    leg = ["Selection","Insertion","Quick","Merge","Bubble"]
    #Orden inverso
    if(indicador == 0):
        revSamp = inversedSamples(orderedSamples(n,floats))
        #Hacemos una copia de los elementos en orden inverso
        #selection,insertion,quick,merge,bubble en ese orden
        #Cada entrada tiene los mismos arreglos pero uno apra cada uno
        Copias = copia(revSamp)
        title = "Orden inverso"


        for i in range(5):
            size,time,comp = Ordena(Metodos[i],Copias[i])
            Size.append(size)
            Time.append(time)
            Comp.append(comp)

    #Orden aleatorio
    elif(indicador == 1):
        title = "Orden Aleatorio"
        Samples = randomSamples(n,floats)
        Copias = copia(Samples)
        for i in range(5):
            size,time,comp = Ordena(Metodos[i],Copias[i])
            Size.append(size)
            Time.append(time)
            Comp.append(comp)

    elif(indicador == 2):
        title = "Datos Ordenados"
```

```
        Samples = orderedSamples(n,floats)
        for metodo in Metodos:
            size,time,comp = Ordena(metodo,Samples)
            Size.append(size)
            Time.append(time)
            Comp.append(comp)

    figure = plt.figure()
    figure.suptitle(title, fontsize=16)


    #Grafico izquierdo
    ax = plt.subplot(1,2,1)
    for i in range(5):
        ax.plot(Size[i],np.log10(Comp[i]))

    ax.set_xlabel("Número de elementos")
    ax.set_ylabel("Log10 Comparaciones")
    ax.grid()
    ax.legend(leg)

    ax1 = plt.subplot(1,2,2)
    for i in range(5):
        ax1.plot(Size[i],np.log10(Time[i]))
    ax1.set_xlabel("Número de elementos")
    ax1.set_ylabel("Log10 Tiempo-Segundos")
    ax1.grid()
    ax1.legend(leg)

#Realiza
def graficaString(n):
    samples = randomStringSample(n)
    Metodos = [selectionSort,insertionSort,quickSort,mergeSort,bubbleSort]
    Size = []
    Time = []
    Comp = []
    leg = ["Selection","Insertion","Quick","Merge","Bubble"]
    Copias = copia(samples)
    for i in range(5):
            size,time,comp = Ordena(Metodos[i],Copias[i])
            Size.append(size)
            Time.append(time)
            Comp.append(comp)

    figure = plt.figure()
    figure.suptitle("Aleatorio Strings", fontsize=16)


    #Grafico izquierdo
    ax = plt.subplot(1,2,1)
    for i in range(5):
        ax.plot(Size[i],np.log10(Comp[i]))

    ax.set_xlabel("Número de elementos")
    ax.set_ylabel("Log10 Comparaciones")
    ax.grid()
    ax.legend(leg)
```

```
    ax1 = plt.subplot(1,2,2)
    for i in range(5):
        ax1.plot(Size[i],np.log10(Time[i]))
    ax1.set_xlabel("Número de elementos")
    ax1.set_ylabel("Log10 Tiempo-Segundos")
    ax1.grid()
    ax1.legend(leg)

"""
Consideraciones:
    -Actualmente la base es 3, todos los métodos, sin importar
    el orden o tipo de dato pueden ordenar arreglos de tamaño hasta
    3^9, por lo menos en la máquina donde fueron probados, a excepción
    del método quickSort en orden inverso, solo soporta hasta
    3^7.

    -La ejecución, según el método y el orden pueden tardar si se realizan
    todas al mismo tiempo
"""

#Individuales
graficaSolo(selectionSort, 0, 9)
graficaSolo(selectionSort, 1, 9)
graficaSolo(selectionSort, 2, 9)

graficaSolo(insertionSort, 0, 9)
graficaSolo(insertionSort, 1, 9)
graficaSolo(insertionSort, 2, 9)

graficaSolo(mergeSort, 0, 9)
graficaSolo(mergeSort, 1, 9)
graficaSolo(mergeSort, 2, 9)

# conjuntos
graficaConjunto(0,7)
graficaConjunto(1,9)
graficaConjunto(2,9)
```