```python
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  1 01:09:34 2021

@author: Guillermo Arredondo
Abraham Guerrero
Luisa Saloma
"""

from Algoritmos import *
import time
import random
import string

"""
pruebas metodo mezcla
F=[0, 10, 12, 13, 24,55]
G=[1, 5, 17, 50, 70]
mezcla(F,G)
mezcla(G,F)
"""

"""Se crearon dos métodos para facilitar la llamada y toma de tiempo entre los algoritmos
efectuados"""
"""
el metodo regresaTiempo recibe un arreglo y el nombre del metodo a probar el cual regresa
la diferencia de tiempo entre el inicio del algoritmo y el final.
"""
def regresaTiempo(Arre,metodo):
  tiempoIn = time.time()
  metodo(Arre)
  tiempoFin = time.time()
  print(metodo,"\n", Arre)
  print(tiempoFin-tiempoIn)

"""El método mostrado a continuación copia la lista/arreglo dos veces más de la original
para poder hacer la llamada de los métodos por separados desde un mismo punto de inicio,
así se verifica su rendimiento con parámetros iguales y no con arreglos ya ordenados.
Asimismo, es importante notar que llama a la función anterior para medir el tiempo de
cada uno de los métodos"""
def compara(Arre):
  Aux1=Arre[0:]
  Aux2=Arre[0:]
  regresaTiempo(Arre,seleccionDirecta)
  regresaTiempo(Aux1,insercionDirecta)
  regresaTiempo(Aux2,combinacionSubColecc)

'''
"""A continuación se muestran las pruebas unitarias por métodos, para evaluar el
correcto funcionamiento de cada uno en arreglos iguales y arbitrarios; del mismo modo,
hace uso de la función compara() para evaluar el arreglo en los tres métodos
Entre las pruebas unitarias se encuentran las básicas como arreglos vacíos,
y con datos pseudo aleatorios de autoría propia
"""

"""A es un arreglo con valores arbitrarios que sirvio para la primera prueba
de todos los algoritmos"""
```

```
A=[6,5,3,1,8,7,2,4]
print("Coleccion enteros A\n",A)
"""
seleccionDirecta(A)
insercionDirecta(A)
combinacionSubColecc(A)
"""

compara(A)

#B es un arreglo con la misma funcion de A pero para intentar mas pruebas
B=[9,8,1,10,17,5,28,30]
print("Coleccion enteros B\n",B)
"""
seleccionDirecta(B)
insercionDirecta(B)
combinacionSubColecc(B)
"""
compara(B)


D=[9,8,1,10,17,5,28,30]
print("Coleccion D\n",D)
"""
seleccionDirecta(D)
insercionDirecta(D)
combinacionSubColecc(D)
"""
compara(D)

"""El arreglo H esta compuesto de char´s donde nos permitio comprobar que
nuestros algoritmos funciona con otros tipos de datos"""

H=["h","g","u","i","l","l","e","r","m","o"]
print("Coleccion caracteres H\n",H)
"""
seleccionDirecta(H)
insercionDirecta(H)
combinacionSubColecc(H)
"""
compara(H)

"""El arreglo S esta formado de strings donde nos permitio comprobar
que nuestros algoritmos funciona con otros tipos de datos"""

S=["hello","ga","un","iglu","llamado","llamada","en","r","ma","sa"]
print("Coleccion cadenas S\n",S)
"""
seleccionDirecta(S)
insercionDirecta(S)
combinacionSubColecc(S)
"""
compara(S)

"""El arreglo M esta formado por diferentes tipos de valores numericos
donde nos permitio comprobar que nuestros algoritmos funciona con
```

```
tipos de datos diferentes entre si mientras sean comparables entre
ellos"""

M=[1, 2.6, 50, 69, -4, -2.5,-50]
print("Coleccion entre float´s M\n",M)
"""
seleccionDirecta(M)
insercionDirecta(M)
combinacionSubColecc(M)
"""
compara(M)

#N es el caso para cuando el arreglo esta vacio
N=[]
print("Coleccion vacia N\n", N)
"""
seleccionDirecta(N)
insercionDirecta(N)
combinacionSubColecc(N)
"""
compara(N)


#U es el caso para cuando el arreglo esta contenido por un solo dato

U=[1]
print("Coleccion con un elemento U\n",U)
"""
seleccionDirecta(U)
insercionDirecta(U)
combinacionSubColecc(U)
"""
compara(U)
'''

"""Para mejorar el uso de arreglos y listas aleatorias, se diseñaron
los siguientes algortimos que en un inicio crearon arreglos con datos
enteros y posteriormente se usaron para rellenar con datos de cadenas,
nótese que para las cadenas se utilizó un ciclo que concatena
caracteres aleatorios por parte de la libreria random"""
def creaListaAleatoria(n):
  A=[]
  for i in range(n):
    #A.append(random.randint(-100, 100))
    m=random.randint(0,10)
    cadena = ""
    for k in range(m):
      cadena+=random.choice("qwedsazxcvfrtgbhnyujmklpoi")
    A.append(cadena)
    #A.append(random.choice("qwedsazxcvfrtgbhnyujmklpoi"))
  print(A)
  return A

"""Para verificar los casos de arreglos ordenados ya sea de forma
ascendente o descendente, se diseñó el siguiente algoritmo que toma
como bandera booleana:
    True en caso de ser ascendente,*
```

```
    False en caso de ser descendente*
  *para evitar un decrecimiento estricto de datos, se diseñó un
  algoritmo que o bien repita el  dato pasado o decremente/incremente en uno"""
def creaListaOrdenada(n, ascendente):
  A=[]
  if(ascendente == True):
    for x in range(n):
      A.append(random.randint(x-1,x))
  else:
    for x in range(n,0,-1):
      A.append(random.randint(x-1,x))
  print(A)
  return A

print("Aleatorio empieza\n\n Seleccion Directa")


"""Originalmente se efectuaron pruebas específicas por cada uno de los
métodos, pidiendo a través de un for que cree 10 (en un inicio) arreglos
de distintas longitudes con los siguientes parámetros:
    1°: range(10)
        random. randint(2,100)
    2°: range(2)
        random.randint(5000,10000)
    3°: range(2)
    random.randint(2000,20000)
nótese que en los segundos casos se utilizó el método compara para
compensar la reducción de número de arreglos creados"""
for i in range(2): #10
  n=random.randint(5000,10000)
  print(n)
  A=creaListaAleatoria(n)
  '''tiempoIn = time.time()
  seleccionDirecta(A)
  tiempoFin = time.time()
  print(tiempoFin-tiempoIn)
  print("Ordenado\n", A)'''
  compara(A)


print("\nInsercion Directa")
for i in range(2):
  n=random.randint(2000,20000)
  print(n)
  A=creaListaAleatoria(n)
  '''tiempoIn = time.time()
  insercionDirecta(A)
  tiempoFin = time.time()
  print(tiempoFin-tiempoIn)
  print("Ordenado\n", A)'''
  compara(A)

print("\nMerge Sort")
B=[]
for i in range(2):
  n=random.randint(5000,10000)#(2,100)
  print(n)
```

```
  A=creaListaAleatoria(n)
  '''tiempoIn = time.time()
  combinacionSubColecc(A)
  tiempoFin = time.time()'''
  compara(A)
  B.append(A)
  #print(tiempoFin-tiempoIn)
  #print("OrdenadoCombinacion\n", A)

"""
Para resolver el reto de la creación de arreglos de cadenas
pseudo ordenadas (con longitud mayor a 1 caracter) se decidió
que por cada lista creada aleatoriamente, posteriormente de
ser ordenada se insertarán en una lista que acumulara las
listas creadas aleatorias (ya ordenadas) para poder recuperarlas
en el método siguiente, el cual llama a Compara() con el
arreglo ya ordenado
"""
print("\n Cadenas ordenadas ascendente")
for lista in B:
  print(len(lista))
  #print(lista)
  compara(lista)


print("\n Enteros Ordenados ascendente")
for x in range(2):
  n=random.randint(5000,10000) #(2,100)
  print(n)
  O=creaListaOrdenada(n, True)
  compara(O)

"""El siguiente método hace uso de una funcion auxiliar
llamada reversed() el cual invierte el orden de los datos de
una coleccion recibida como paramétro, en este caso se utilizó
para reordenar las listas de arreglos de forma descendente,
fue una herramienta muy útil debido a la dificultad de generar
listas de cadenas ordenadas con muchos datos de una forma automática
"""
print("\n Cadenas ordenadas descendente")
for lista in B:
  listaInv= list(reversed(lista))
  print(len(listaInv))
  #print(listaInv)
  compara(listaInv)

print("\n Enteros Ordenados descendente")
for x in range(2):
  n=random.randint(5000,10000) #(2,100)
  print(n)
  O=creaListaOrdenada(n,False)
  compara(O)
```