Ramírez González Leonardo Ramses #17211554

Prueba de velocidad de métodos ordenamiento (Bubblesort, Quicksort, Shellsort y Mergesort).

Para probar la eficiencia de los métodos de ordenamiento se generó un arreglo de 2500 elementos diferentes. Y se sometió a cada método de ordenamiento a ordenar dicho arreglo un total de 50 veces cada método.
Los resultados son los siguientes:

```
Bubblesort:
Test 1: 1.9992
Test 2: 1.9097
Test 3: 1.8148
Test 4: 1.7232
Test 5: 1.7432
Test 6: 1.9002
Test 7: 1.7945
Test 8: 1.9038
Test 9: 1.8766
Test 10: 1.8339
Test 11: 1.8825
Test 12: 1.7809
Test 13: 2.3291
Test 14: 1.9152
Test 15: 1.7276
Test 16: 1.6201
Test 17: 1.5836
Test 18: 1.7701
Test 19: 1.8233
Test 20: 1.7271
Test 21: 1.7472
Test 22: 1.8175
Test 23: 1.7292
Test 24: 1.7543
Test 25: 1.7916
Test 26: 1.8139
Test 27: 1.7318
Test 28: 1.7667
Test 29: 1.8009
Test 30: 1.7447
Test 31: 1.8463
Test 32: 1.7721
Test 33: 1.8383
Test 34: 1.88
Test 35: 1.8759
Test 36: 1.9629
Test 37: 1.892
Test 38: 2.0141
Test 39: 1.8904
Test 40: 1.8803
Test 41: 1.7428
Test 42: 1.7642
Test 43: 1.8535
Test 44: 1.7037
Test 45: 1.792
Test 46: 1.7819
Test 47: 1.8696
Test 48: 1.8467
Test 49: 1.7184
Test 50: 1.8564
        Promedio: 1.8228 s
```

```
Quicksort:
Test 1: 0.0185
Test 2: 0.0146
Test 3: 0.0224
Test 4: 0.0224
Test 5: 0.0107
Test 6: 0.0185
Test 7: 0.0195
Test 8: 0.021
Test 9: 0.0234
Test 10: 0.0195
Test 11: 0.0269
Test 12: 0.0185
Test 13: 0.0195
Test 14: 0.0332
Test 15: 0.0107
Test 16: 0.0215
Test 17: 0.0215
Test 18: 0.0342
Test 19: 0.023
Test 20: 0.0234
Test 21: 0.0068
Test 22: 0.0224
Test 23: 0.0156
Test 24: 0.0127
Test 25: 0.0264
Test 26: 0.0224
Test 27: 0.0185
Test 28: 0.0244
Test 29: 0.0166
Test 30: 0.0146
Test 31: 0.0195
Test 32: 0.0215
Test 33: 0.0166
Test 34: 0.0146
Test 35: 0.0195
Test 36: 0.0176
Test 37: 0.0371
Test 38: 0.0283
Test 39: 0.0283
Test 40: 0.0195
Test 41: 0.0244
Test 42: 0.0273
Test 43: 0.0107
Test 44: 0.0273
Test 45: 0.0234
Test 46: 0.0205
Test 47: 0.0234
Test 48: 0.0176
Test 49: 0.0176
Test 50: 0.0195
        Promedio: 0.0207 s
```

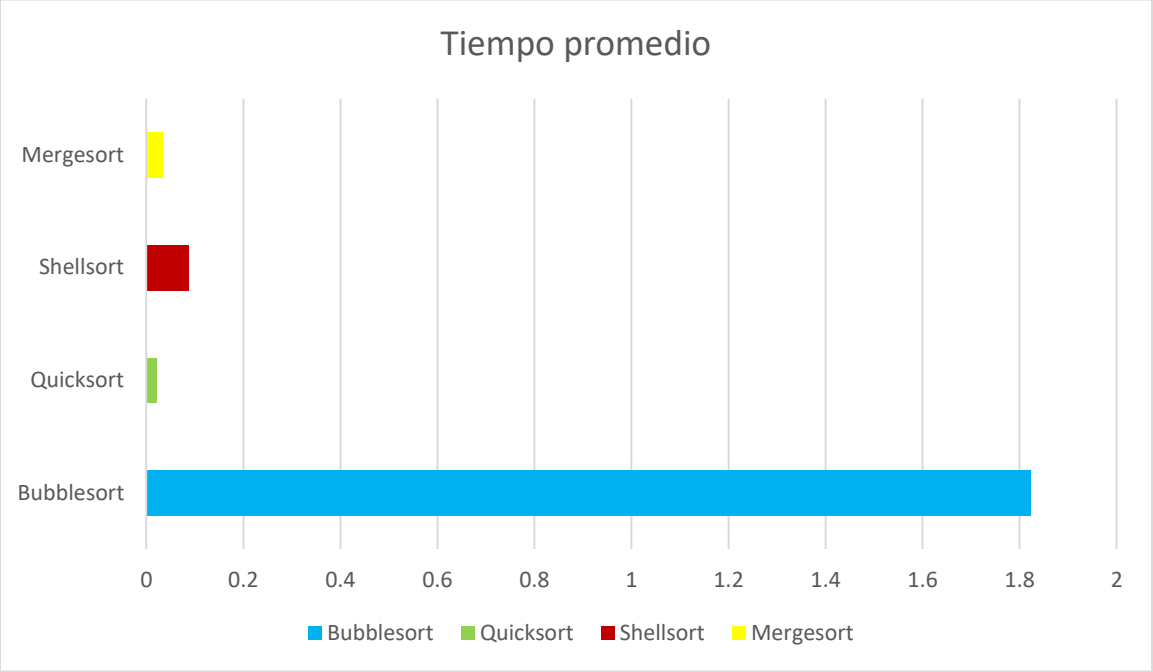
```
Shellsort:
Test 1: 0.0996
Test 2: 0.1054
Test 3: 0.0732
Test 4: 0.0937
Test 5: 0.0751
Test 6: 0.0654
Test 7: 0.0586
Test 8: 0.0761
Test 9: 0.0742
Test 10: 0.1049
Test 11: 0.0927
Test 12: 0.0986
Test 13: 0.0898
Test 14: 0.1122
Test 15: 0.0908
Test 16: 0.0981
Test 17: 0.1118
Test 18: 0.0815
Test 19: 0.0923
Test 20: 0.1079
Test 21: 0.1044
Test 22: 0.0742
Test 23: 0.0927
Test 24: 0.0703
Test 25: 0.081
Test 26: 0.0937
Test 27: 0.0937
Test 28: 0.0952
Test 29: 0.08
Test 30: 0.1015
Test 31: 0.0781
Test 32: 0.083
Test 33: 0.0732
Test 34: 0.0966
Test 35: 0.0864
Test 36: 0.0893
Test 37: 0.0947
Test 38: 0.0673
Test 39: 0.0547
Test 40: 0.0908
Test 41: 0.0976
Test 42: 0.0966
Test 43: 0.0825
Test 44: 0.0693
Test 45: 0.08
Test 46: 0.0956
Test 47: 0.1054
Test 48: 0.0883
Test 49: 0.0971
Test 50: 0.0981
        Promedio: 0.0883 s
```

```
Mergesort:
Test 1: 0.0361
Test 2: 0.0361
Test 3: 0.0429
Test 4: 0.04
Test 5: 0.0312
Test 6: 0.0449
Test 7: 0.0254
Test 8: 0.042
Test 9: 0.0351
Test 10: 0.039
Test 11: 0.0322
Test 12: 0.0263
Test 13: 0.039
Test 14: 0.0371
Test 15: 0.0342
Test 16: 0.0342
Test 17: 0.0293
Test 18: 0.0361
Test 19: 0.0371
Test 20: 0.0371
Test 21: 0.0283
Test 22: 0.0429
Test 23: 0.0351
Test 24: 0.0312
Test 25: 0.0332
Test 26: 0.0371
Test 27: 0.0351
Test 28: 0.0264
Test 29: 0.0278
Test 30: 0.0254
Test 31: 0.0342
Test 32: 0.0225
Test 33: 0.0322
Test 34: 0.0234
Test 35: 0.0342
Test 36: 0.0342
Test 37: 0.0293
Test 38: 0.0342
Test 39: 0.0293
Test 40: 0.0332
Test 41: 0.0371
Test 42: 0.0215
Test 43: 0.0371
Test 44: 0.0205
Test 45: 0.0429
Test 46: 0.0215
Test 47: 0.0342
Test 48: 0.0361
Test 49: 0.0342
Test 50: 0.0332
        Promedio: 0.0333 s
```

Tabla comparativa de promedios:

Método de ordenamiento	Promedio de tiempo
Bubblesort	1.8228 segundos
Quicksort	0.0207 segundos
Shellsort	0.0883 segundos
Mergesort	0.0333 segundos

Como se puede observar quicksort es el metodo más rapido en esta prueba con una diferencia de 0.0126 segundos en comparacion al mergesort. Y el bubblesort el más lento quedando 1.8001 segundos atrás de quicksort.



Se realizó una segunda prueba usando 10,000 elementos y se repitió 10 veces con cada método. Los resultados obtenidos fueron los siguientes:

```
Bubblesort:
Test 1: 11.302
Test 2: 11.1707
Test 3: 11.8811
Test 4: 12.2404
Test 5: 11.2978
Test 6: 11.0956
Test 7: 10.956
Test 8: 10.7364
Test 9: 11.1083
Test 10: 11.5623
        Promedio: 11.3351 s

******************************
Quicksort:
Test 1: 0.0429
Test 2: 0.0439
Test 3: 0.0399
Test 4: 0.0459
Test 5: 0.0349
Test 6: 0.0369
Test 7: 0.0379
Test 8: 0.0389
Test 9: 0.0409
Test 10: 0.0419
        Promedio: 0.0404 s
```

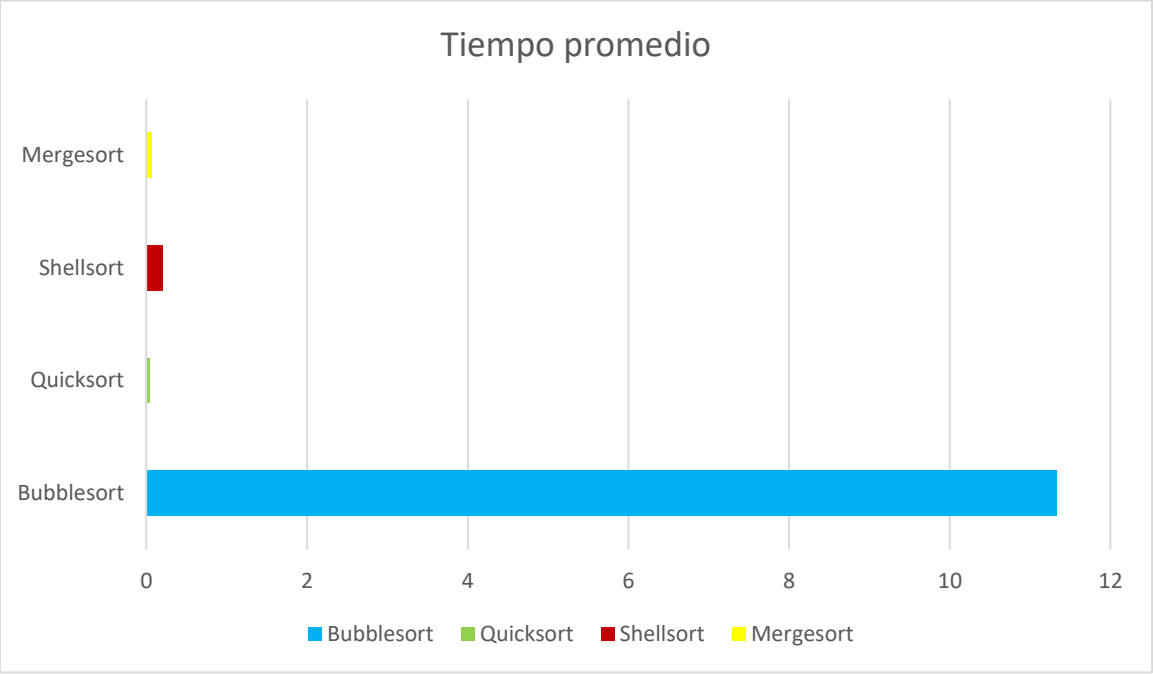
```
Shellsort:
Test 1: 0.1915
Test 2: 0.2565
Test 3: 0.2075
Test 4: 0.2414
Test 5: 0.2026
Test 6: 0.2015
Test 7: 0.2026
Test 8: 0.1925
Test 9: 0.1928
Test 10: 0.1896
        Promedio: 0.2079 s

******************************
Mergesort:
Test 1: 0.0718
Test 2: 0.0669
Test 3: 0.0668
Test 4: 0.0678
Test 5: 0.0678
Test 6: 0.0658
Test 7: 0.0679
Test 8: 0.0648
Test 9: 0.0668
Test 10: 0.0673
        Promedio: 0.0674 s
```

Tabla de promedios:

Método de ordenamiento	Promedio de tiempo
Bubblesort	11.3351 segundos
Quicksort	0.0404 segundos
Shellsort	0.2079 segundos
Mergesort	0.0674 segundos

Se vuelve a demostrar que el quicksort es por mucho muy superior a los demas quedando 0.027 segundos distante del segundo puesto, mergesort. El metodo bubblesort quedo por los suelos tardando 11.3351 segundos, siendo 280 veces más rapido el metodo quicksort, una verdadera barbaridad.



El codigo utilizado para estas pruebas fue el siguiente:

```
from random import shuffle
import copy
import time
#####################################Ordenamiento Burbuja#####################################
def burbuja(miArreglo):
    for i in range (0, len(miArreglo)-1):
        for j in range (0, len(miArreglo)-1-i):
            if (miArreglo[j] > miArreglo[j+1]):
                miArreglo[j], miArreglo[j+1] = miArreglo[j+1], miArreglo[j]

#####################################Ordenamiento quicksort#####################################
def quicksort(arreglo):
    #Si la longitud del arreglo recivido es mayor a 1, dale
    if len(arreglo) > 1:
        #Declarar los tres arreglos a usar
        arrMayores = []
        arrMenores = []
        arrMedios = []
        #Asignamos el pivote en el primer elemento
        piv = arreglo[0]

        #Comparamos los elementos con el pivote
        #Asignamos a cada arreglo los menores/mayores
        #Respecto a nuestro pivote
        for i in range(0, len(arreglo)):
            if arreglo[i] < piv:
                arrMenores.append(arreglo[i])
            elif  arreglo[i] > piv:
                arrMayores.append(arreglo[i])
            else:
                arrMedios.append(arreglo[i])

        #Retornamos los tres arreglos concatenados
        return quicksort(arrMenores) + arrMedios + quicksort(arrMayores)
    else:
        #En caso de el arreglo solo tener un elemento
        #retornamos el mismo arreglo sin alterarlo
        return arreglo

#####################################Ordenamiento shelsort#####################################
def shellsort(miArreglo):
    reco = len(miArreglo)//2

    #Hasta que reco sera 0.
    while(reco!=0):
        #Decimos que hay cambio para entrar al ciclo
        cambio = 1
        #Mientras haya cambio, seguiremos iterando
        while cambio == 1:
            cambio = 0
            #Comparamos los elementos usando el reco para tomarlos.
            for index in range(0,len(miArreglo)-reco):
                #Si es mayor, se intercambian
                if (miArreglo[index] > miArreglo[index+reco]):
                    miArreglo[index],miArreglo[index+reco]=miArreglo[index+reco],miArreglo[index]
                    cambio = 1
        #Dividimos nuevamente el reco
        reco = reco//2
```

```
#####################################Ordenamiento mergesort#######################################
def mergesort(ahri):
    #Si el arreglo tiene mas de 1 elemento entraremos al proceso
    if (len(ahri)>1):
        med = len(ahri)//2

        #Partimos en mitades hasta llegar a 1 elemento por arreglo
        ahriLeft = mergesort(ahri[:med])
        ahriRight = mergesort(ahri[med:])

        #Empezamos a comparar los elementos y los vamos ingresando al nuevo arreglo
        ahriOrd = []

        while(len(ahriLeft) != 0 and len(ahriRight) != 0):
            if ahriLeft[0]< ahriRight[0]: #Ingresamos del arreglo correspondiente al arreglo ordena
               ahriOrd.append(ahriLeft[0]) #Y quitamos del arreglo que corresponda
               ahriLeft.remove(ahriLeft[0])
            else:
                ahriOrd.append(ahriRight[0])
                ahriRight.remove(ahriRight[0])

        #Para evitar problemas de desbordamiento, usamos una comparacion fuera del ciclo
        #Ya que puede darse el caso de que sobre un elemento en alguno de los dos arreglos
        #De ser asi lo agregamos al arreglo.
        if len(ahriLeft) !=0:
            ahriOrd += ahriLeft
        else:
            ahriOrd += ahriRight

        return ahriOrd #Y retornamos el arreglo ya ordenado
    else:
        return (ahri) #En caso de que el arreglo sea de 1 elemento, lo retornamos tal cual llego
```

```
#############################################################################################################

rep = 50 #Cuantas repeticiones queremos probar
numElem = 2500 #Numero de elementos que se usaran en el arreglo

arregloPrincipal = [[i] for i in range(numElem)] #Generar un arreglo de numElem elementos
shuffle(arregloPrincipal) #Revolvemos el arreglo generado antes

tiempoInicio = 0 #Marca el inicio del programa
tiempoTest = 0 #Es el total calculado desde el inicio hasta que se detuvo el programa
tiempoPromedio = 0 #El promedio de todos los tiempos capturados del metodo

print("Bubblesort: ")
for i in range(0,rep): #Iteramos el numero dado de repeticiones
    arr = arregloPrincipal.copy() #Cargamos el metodo generado a un auxiliar
    tiempoInicio = time.time() #tomamos el tiempo inicial
    burbuja(arr) #Invocamos al metodo
    tiempoTest = float("{0:.4f}".format(time.time() - tiempoInicio)) #Capturamos el tiempo del test
    print("Test "+str(i+1)+": "+str(tiempoTest)) #imprimimos el tiempo medido en i prueba
    tiempoPromedio += tiempoTest #Sumaremos cada tiempo que se mida
tiempoPromedio = tiempoPromedio / rep #Promediamos los tiempos obtenidos
print("\tPromedio: "+"{0:.4f}".format(tiempoPromedio)+" s") #Desplegamos el promedio


print("\n*****************************************\nQuicksort: ")
tiempoPromedio = 0 #Reiniciamos el promedio
for i in range(0,rep): #Iteramos el numero dado de repeticiones
    arr = arregloPrincipal.copy() #Cargamos el metodo generado a un auxiliar
    tiempoInicio = time.time() #tomamos el tiempo inicial
    quicksort(arr) #Invocamos al metodo
    tiempoTest = float("{0:.4f}".format(time.time() - tiempoInicio)) #Capturamos el tiempo del test
    print("Test "+str(i+1)+": "+str(tiempoTest)) #imprimimos el tiempo medido en i prueba
    tiempoPromedio += tiempoTest #Sumaremos cada tiempo que se mida
tiempoPromedio = tiempoPromedio / rep #Promediamos los tiempos obtenidos
print("\tPromedio: "+"{0:.4f}".format(tiempoPromedio)+" s") #Desplegamos el promedio


print("\n*****************************************\nShellsort: ")
tiempoPromedio = 0 #Reiniciamos el promedio
for i in range(0,rep): #Iteramos el numero dado de repeticiones
    arr = arregloPrincipal.copy() #Cargamos el metodo generado a un auxiliar
    tiempoInicio = time.time() #tomamos el tiempo inicial
    shellsort(arr) #Invocamos al metodo
    tiempoTest = float("{0:.4f}".format(time.time() - tiempoInicio)) #Capturamos el tiempo del test
    print("Test "+str(i+1)+": "+str(tiempoTest)) #imprimimos el tiempo medido en i prueba
    tiempoPromedio += tiempoTest #Sumaremos cada tiempo que se mida
tiempoPromedio = tiempoPromedio / rep #Promediamos los tiempos obtenidos
print("\tPromedio: "+"{0:.4f}".format(tiempoPromedio)+" s") #Desplegamos el promedio



print("\n*****************************************\nMergesort: ")
tiempoPromedio = 0 #Reiniciamos el promedio
for i in range(0,rep): #Iteramos el numero dado de repeticiones
    arr = arregloPrincipal.copy() #Cargamos el metodo generado a un auxiliar
    tiempoInicio = time.time() #tomamos el tiempo inicial
    mergesort(arr) #Invocamos al metodo
    tiempoTest = float("{0:.4f}".format(time.time() - tiempoInicio)) #Capturamos el tiempo del test
    print("Test "+str(i+1)+": "+str(tiempoTest)) #imprimimos el tiempo medido en i prueba
    tiempoPromedio += tiempoTest #Sumaremos cada tiempo que se mida
tiempoPromedio = tiempoPromedio / rep #Promediamos los tiempos obtenidos
print("\tPromedio: "+"{0:.4f}".format(tiempoPromedio)+" s") #Desplegamos el promedio
```

Proyecto en Github:

https://github.com/RamirezNOD/EstructurasNOD/blob/master/ProyectoFinal.py