	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: MARCO ANTONIO MARTINEZ QUINTANA

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS I

Grupo: 17

No de Práctica(s): 10

Integrante(s): José Luis Arroyo Chavarría

No. de Equipo de cómputo empleado: 1

No. de Lista o Brigada: 5

Semestre: 2

Fecha de entrega: 07/04/2020

Observaciones:

CALIFICACIÓN: _____

Objetivo:

Dar varios enfoques de diseño de algoritmos y analizar las implicaciones de cada uno de ellos.

Introducción:

- **Fuerza bruta:**

El objetivo de resolver problemas por medio de fuerza es bruta es hacer una búsqueda exhaustiva de todas las posibilidades que lleven a la solución del problema.

Un ejemplo de esto es encontrar una contraseña haciendo una combinación exhaustiva y tardada en los caracteres alfanuméricos generando cadenas de cierta longitud.

El uso de la biblioteca string, de ésta se va a importar los caracteres y dígitos.

La biblioteca itertools tiene una función llamada product() la cual se va a utilizar para realizar las combinaciones en cadenas de hasta 4 caracteres.

Las combinaciones generadas por el algoritmo se van a guardar en un archivo.

Para guardar datos en un archivo se utiliza la función open(), es para tener una referencia del archivo que se quiere abrir. La función write(), que recibe la cadena que se va a escribir en el archivo. Finalmente, una vez que se termina la escritura hacia el archivo, éste se cierra con la función close().

- **Algoritmos ávidos (greedy):**

Esta estrategia toma una serie de decisiones en un orden específico, una vez que se ha ejecutado esa decisión, ya no se vuelve a considerar. Una desventaja es que la solución que se obtiene no siempre es la más óptima.

Un ejemplo es al utilizar el operando //. La diferencia entre utilizar / y // es que el primer operador realiza una operación de números reales y el segundo una de números enteros.

5/2 = 2.5 5//2 = 2

- **Bottom-up (programación dinámica):**

El objetivo de esta estrategia es resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular

las soluciones.

Como ejemplo, se va a calcular el número n de la sucesión de Fibonacci. La sucesión de Fibonacci es una sucesión infinita de números enteros cuyos primeros dos elementos son 0 y 1, los siguientes números son calculados por la suma de los dos anteriores.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Al aplicar la estrategia bottom-up. Partimos del hecho de que ya tenemos las soluciones. Se toman las soluciones ya existentes. La solución se encuentra calculando los resultados desde los casos base de abajo hacia arriba.

- **Top-down:**

Se empiezan a hacer los cálculos hacia abajo. Además, se aplica una técnica memorización, consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones.

Para aplicar la estrategia top-down, se utiliza un diccionario (memoria) el cual va a almacenar valores previamente calculados. Una vez que se realice el cálculo de algún elemento, éste se va a almacenar ahí.

Como se muestra en la impresión de la variable memoria, que contiene los resultados previamente calculados, los nuevos valores obtenidos se agregaron a ésta.

Se requiere que los valores ya calculados sean guardados en un archivo, de tal manera que se puedan utilizar en otro instante de tiempo. Para esto se va a hacer uso de la biblioteca pickle. Los archivos que se generan con pickle están en binario, por lo que no se puede leer a simple vista la información que contienen, como se haría desde un archivo de texto plano.

- **Incremental:**

Es una estrategia que consiste en implementar y probar que sea correcto de manera paulatina, ya que en cada iteración se va agregando información.

❖ *Insertion sort*

Ordena los elementos manteniendo una sublista de números ordenados empezando por las primeras localidades de la lista. Al principio se considera que el elemento en la primera posición de la lista está ordenado. Después cada uno de los

elementos de la lista se compara con la sublista ordenada para encontrar la posición adecuada.

- **Divide y vencerás:**

Divide el problema en subproblemas hasta que son suficientemente simples que se pueden resolver directamente.

Después las soluciones son combinadas para generar la solución general del problema.

❖ *Quick sort*

Resuelve un problema por medio de la estrategia divide y vencerás. En Quicksort se divide en dos el arreglo que va a ser ordenado y se llama recursivamente para ordenar las divisiones. Quicksort es la partición de los datos. Es escoger un valor de pivote el cual está encargado de ayudar con la partición de los datos. El objetivo de dividir los datos es mover los que se encuentran en una posición incorrecta con respecto al pivote.

- **Medición y gráficas de los tiempos de ejecución:**

En dado caso de que se quiera llamar más funciones que estén en un mismo archivo se pueden escribir los nombres de las funciones separados por nombres: *from file_name import función1, función2, función3*

- **Modelo RAM**

Cuando se realiza un análisis de complejidad utilizando el modelo RAM, se debe contabilizar las veces que se ejecuta una función o un ciclo, en lugar de medir el tiempo de ejecución.

Desarrollo y resultados:

- **Código**

1. **Fuerza bruta**

```
from string import ascii_letters , digits
from itertools import product

#Concatenar letras y digitos en una sola cadena
caracteres = ascii_letters+digits
```

```
def buscador(con)

    #Archivo con todas las combinaciones generadas
    archivo = open ("combinaciones.txt", "w")

    if 3<= len(con) <= 4:
        for i in range(3,5):
            for comb in product(caracteres, repeat = i):
                #Se utiliza join() para concatenar los caracteres regresando por la
funcion product().
                #Como join necesita una cadena inicial para hacer la
concatenacion, se pone una cadena vacia
                #al principio
                prueba = "".join(comb)
                #Escribiendo al archivo cada combinacion generada
                archivo.write( prueba + "\n" )
                if prueba == con:
                    print('Tu contraseña es {}'.format(prueba))
                    #Cerrando el archivo
                    archivo.close()
                    break
    else:
        print('Ingresa una contraseña que contenga de 3 a 4 caracteres')

from time import time
tO = time()
con = 'H0l4'
buscador(con)
print("Tiempos de ejecucion {}".format(round(time()-tO, 6)))
```

2. Algoritmos ávidos (greedy)

```
def cambio(cantidad, denominaciones):
```

```
    resultado = []
    while (cantidad > 0):
        if(cantidad >= denominaciones[0]):

            num = cantidad // denominaciones[0]
            cantidad = cantidad - (num * denominaciones[0])
            resultado.append([denominaciones[0], num])
        denominaciones = denominaciones[1:] #Se va consumiendo la lista de denominaciones
    return resultado

#Pruebas del algoritmo
print (cambio(1000, [500, 200, 100, 50, 20, 5, 1]))

print (cambio(500, [500, 200, 100, 50, 20, 5, 1]))

print (cambio(300, [50, 20, 5, 1]))

print (cambio(200, [5]))

print (cambio(98, [50, 20, 5, 1]))

print (cambio(98, [5, 20, 1, 50]))
```

3. Bottom-up (programación dinámica)

-
```
def fibonacci_iterativo_v1(numero):
    f1=0
    f2=1
    tmp=0
    for i in range(1, numero-1):
        tmp = f1+f2
        f1=f2
```

```
        f2=tmp
    return f2
fibonacci_iterativo_v1(6)
```

- ```
  def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2
    return f2
  fibonacci_iterativo_v2(6)
  ```
- ```
  def fibonacci_bottom_up(numero):
    f_parciales = [0, 1, 1] #Esta es la lista que mantiene las soluciones previamente calculadas
    while len(f_parciales) < numero:
        f_parciales.append(f_parciales[-1] + f_parciales[-2])
        print(f_parciales)
    return f_parciales[numero-1]
  fibonacci_bottom_up(5)
  ```

4. Top-down

```
def fibonacci_top_down(numero):
    if numero in memoria: #Si el numero ya se encuentra calculando, se regresa el valor ya ya no se hacen mas calculos
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria[numero]
```

5. Pickle

```
#Se carga la biblioteca
import pickle

#Guardar variable
```

```
#No hay restriccion en lo que se pone como extension del archivo,
#generalmente se usa .p o pickle como estandar.
archivo = open('memoria.p', 'wb') #Se abre el archivo para escribir en modo binario
pickle.dump(memoria, archivo) #se guarda la variable memoria que es un diccionario
archivo.close() #se cierra el archivo

#leer variable
archivo = open('memoria.p', 'rb') #Se abre el archivo para leer en modo binario
memoria_de_archivo = pickle.load(archivo) #Se lee la varible
archivo.close() #Se cierra el archivo
```

6. Insertion sort

```
def insertionSort(n_lista):
    for index in range(1,len(n_lista)):
        actual = n_lista[index]
        posicion = index
        print("valor a ordenar = {}".format(actual))
        while posicion>0 and n_lista[posicion-1]>actual:
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
        print(n_lista)
        print()
    return n_lista

#Datos de entrada
lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
insertionSort(lista)
print("lista ordenada {}".format(lista))
```

7. Quick sort

```
def quicksort(lista):
    quicksort_aux(lista,0,len(lista)-1)

def quicksort_aux(lista,inicio, fin):
    if inicio < fin:

        pivote = particion(lista,inicio,fin)

        quicksort_aux(lista, inicio, pivote-1)
        quicksort_aux(lista, pivote+1, fin)

def particion(lista, inicio, fin):
    #Se asigna como pivote en numero de la primera localidad
    pivote = lista[inicio]
    print("Valor del pivote {}".format(pivote))
    #Se crean dos marcadores
    izquierda = inicio+1
    derecha = fin
    print("Indice izquierdo {}".format(izquierda))
    print("Indice derecho {}".format(derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while lista[derecha] >= pivote and derecha >=izquierda:
            derecha = derecha -1
        if derecha < izquierda:
            bandera= True
        else:
            temp=lista[izquierda]
```

```
            lista[izquierda]=lista[derecha]
            lista[derecha]=temp
    print(lista)

    temp=lista[inicio]
    lista[inicio]=lista[derecha]
    lista[derecha]=temp
    return

lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
quicksort(lista)
print("lista ordenada {}".format(lista))
```

8. Medición y gráficas de los tiempos de ejecución

- #Importando bibliotecas

```
%pylab inline
import matplot.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

- #Cargando modulos

```
import random
from time import time

#Cargando las funciones guardadas en los archivo
from insertionSort import insertionSort_time
#Solo se necesita llamar a la funcion principal
from quickSort import quicksort_time
```

- #Tamaños de la lista de numeros aleatorios a generar

```
datos = [ii*100 for ii in range(1,21)]

tiempo_is = [] #Lista para guardar el tiempo de ejecucion de insert sort
```

```
tiempo_qs = [] #Lista para guardar el tiempo de ejecucion de quick sort

for ii in datos:
    lista_is = random.sample(range(0, 10000000), ii)
    #Se hace una copia de la lista para que se ejecute el algoritmo con los mismos numeros
    lista_qs = lista_is.copy()

    tO = time() #Se guarda el tiempo inicial
    quicksort_time(lista_qs)
    tiempo_qs.append(round(time()-tO, 6))
```

- #Generando la grafica

```
fig, ax = subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker"*",color="r")
ax.plot(datos, tiempo_qs, label="quick sort", marker"o",color="b")
ax.set_xlabel('Datos')
ax.set_ylabel('Tiempo')
ax.grid(True)
ax.legend(loc=2)

plt.title('Tiempo de ejecucion [s] (insert vs. quick)')
plt.show[]
```

- #Imprimiendo tiempos parciales de ejecucion

```
print("Tiempos parciales de ejecucion en INSERT SORT {} [s] \n".format(tiempo_is))
print("Tiempos parciales de ejecucion en QUICK SORT {} [s]".format(tiempo_qs))
```

- #Imprimiendo tiempos totales de ejecucion

```
#Para calcular el tiempo total se le aplica la funcion sum() a las listas de tiempo
print("Tiempos totales de ejecucion en INSERT SORT {} [s]".format(sum(tiempo_is))
print("Tiempos totales de ejecucion en QUICK SORT {} [s]".format(sum(tiempo_qs)))
```

9. Modelo RAM

```
import matplot.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D

times = 0

def insertionSort_graph(n_lista):
    global times
    for index in range(1,len(n_lista)):
        times += 1
        actual = n_lista[index]
        posicion = index
        while posicion>0 and n_lista[posicion-1]>actual:
            times += 1
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
    return n_lista
TAM = 101
eje_x = list(range(1,TAM,1))
eje_y = []
lista_variable = []

for num in eje_x:
    lista_variable = ramdom.sample(range(0, 1000), num)
    times = 0
    lista_variable = insertionSort_graph(lista_variable)
    eje_y.append(times)

fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(eje_x, eje_y, marker="o" ,color="b" , linestyle='None')

ax.set_xlabel('x')
```

ax.set_yabel('y')

ax.grid(True)

ax.legend(["Insertion sort"])

plt.title('Insertion sort')

plt.show()

Captura de pantalla

```
from string import ascii_letters , digits
from itertools import product

#Concatenar letas y digitos en una sola cadena
caracteres = ascii_letters+digits

def buscador(con):

    #Archivo con todas las combinaciones generadas
    archivo = open("combinaciones.txt", "w")

    if 3<= len(con) <= 4:
        for i in range(3, 5):
            for comb in product(caracteres, repeat = i):
                #Se utiliza join() para concatenar los caracteres regresando por la funcion product().
                #Como join necesita una cadena inicial para hacer la concatenacion, se pone una cadena vacia
                #al principio
                prueba = "".join(comb)
                #Escribiendo al archivo cada combinacion generada
                archivo.write( prueba + "\n" )
                if prueba == con:
                    print('tu contraseña es {}'.format(prueba))
                    #Cerrando el archivo
                    archivo.close()
                    break
    else:
        print('Ingresa una contraseña que contenga de 3 a 4 caracteres')

from time import time
t0 = time()
con = 'HOl4'
buscador(con)
print("Tiempos de ejecucion {}".format(round(time()-t0, 6)))
```

```
tu contraseña es HOl4
Tiempos de ejecucion 3.605763
```

```
def cambio(cantidad, denominaciones):
    resultado = []
    while (cantidad > 0):
        if(cantidad >= denominaciones[0]):

            num = cantidad // denominaciones[0]
            cantidad = cantidad - (num * denominaciones[0])
            resultado.append([denominaciones[0], num])
        denominaciones = denominaciones[1:] #Se va consumiendo la lista de denominaciones
    return resultado
#Pruebas del algoritmo
print (cambio(1000, [500, 200, 100, 50, 20, 5, 1]))

print (cambio(500, [500, 200, 100, 50, 20, 5, 1]))

print (cambio(300, [50, 20, 5, 1]))

print (cambio(200, [5]))

print (cambio(98, [50, 20, 5, 1]))
```

```
[[500, 2]]
[[500, 1]]
[[50, 6]]
[[5, 40]]
[[50, 1], [20, 2], [5, 1], [1, 3]]
```

```python
def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1,numero-1):
        f1,f2=f2,f1+f2 #Asignacion paralela
    return f2
fibonacci_iterativo_v2(6)
```

```
5
```

```python
def fibonacci_bottom_up(numero):
    f_parciales = [0, 1, 1] #Esta es la lista que mantiene las soluciones previamente calculadas
    while len(f_parciales) < numero:
        f_parciales.append(f_parciales[-1] + f_parciales[-2])
        print(f_parciales)
    return f_parciales[numero-1]
fibonacci_bottom_up(5)
```

```
[0, 1, 1, 2]
[0, 1, 1, 2, 3]
```

```
3
```

```python
#Memoria inicial
memoria = {1:0, 2:1, 3:1}

def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2
    return f2

def fibonacci_top_down(numero):
    if numero in memoria: #Si el numero ya se encuentra calculando, se regresa
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria[numero]

fibonacci_top_down(12)
```

```
89
```

```python
fibonacci_top_down(12)
memoria
```

```
{1: 0, 2: 1, 3: 1, 12: 89}
```

```python
fibonacci_top_down(8)
memoria
```

```
{1: 0, 2: 1, 3: 1, 8: 13}
```

```python
fibonacci_top_down(8)
```

```
13
```

```python
#Memoria inicial
memoria = {1:0, 2:1, 3:1}

def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2
    return f2

def fibonacci_top_down(numero):
    if numero in memoria: #Si el numero ya se encuentra calculando, se regresa e
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria[numero]

#Se carga la biblioteca
import pickle

#Guardar variable
#No hay restriccion en lo que se pone como extension del archivo,
#generalmente se usa .p o pickle como estandar.
archivo = open('memoria.p', 'wb') #Se abre el archivo para escribir en modo bina
pickle.dump(memoria, archivo) #se guarda la variable memoria que es un diccionar
archivo.close() #se cierra el archivo

#Leer variable
archivo = open('memoria.p', 'rb') #Se abre el archivo para leer en modo binario
memoria_de_archivo = pickle.load(archivo) #Se lee la varible
archivo.close() #Se cierra el archivo

memoria_de_archivo
```

```
{1: 0, 2: 1, 3: 1}
```

```python
memoria
```

```
{1: 0, 2: 1, 3: 1}
```

```python
def insertionSort(n_lista):
    for index in range(1,len(n_lista)):
        actual = n_lista[index]
        posicion = index
        print("valor a ordenar = {}".format(actual))
        while posicion>0 and n_lista[posicion-1]>actual:
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
        print(n_lista)
        print()
    return n_lista

#Datos de entrada
lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
insertionSort(lista)
print("lista ordenada {}".format(lista))
```

```
lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
valor a ordenar = 10
[10, 21, 0, 11, 9, 24, 20, 14, 1]

valor a ordenar = 0
[0, 10, 21, 11, 9, 24, 20, 14, 1]

valor a ordenar = 11
[0, 10, 11, 21, 9, 24, 20, 14, 1]

valor a ordenar = 9
[0, 9, 10, 11, 21, 24, 20, 14, 1]

valor a ordenar = 24
[0, 9, 10, 11, 21, 24, 20, 14, 1]

valor a ordenar = 20
[0, 9, 10, 11, 20, 21, 24, 14, 1]

valor a ordenar = 14
[0, 9, 10, 11, 14, 20, 21, 24, 1]

valor a ordenar = 1
[0, 1, 9, 10, 11, 14, 20, 21, 24]

lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
```

```
def quicksort(lista):
    quicksort_aux(lista,0,len(lista)-1)

def quicksort_aux(lista,inicio, fin):
    if inicio < fin:

        pivote = particion(lista,inicio,fin)

        quicksort_aux(lista, inicio, pivote-1)
        quicksort_aux(lista, pivote+1, fin)

def particion(lista, inicio, fin):
    #Se asigno como pivote en numero de la primera localidad
    pivote = lista[inicio]
    print("Valor del pivote {}".format(pivote))
    #Se crean dos marcadores
    izquierda = inicio+1
    derecha = fin
    print("Indice izquierdo {}".format(izquierda))
    print("Indice derecho {}".format(derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while lista[derecha] >= pivote and derecha >=izquierda:
            derecha = derecha -1
        if derecha < izquierda:
            bandera= True
        else:
            temp=lista[izquierda]
            lista[izquierda]=lista[derecha]
            lista[derecha]=temp

    print(lista)

    temp=lista[inicio]
    lista[inicio]=lista[derecha]
    lista[derecha]=temp
    return

lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
quicksort(lista)
print("lista ordenada {}".format(lista))
```

```
lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
Valor del pivote 21
Indice izquierdo 1
Indice derecho 8
[21, 10, 0, 11, 9, 1, 20, 14, 24]
```

```
Valor del pivote 14
Indice izquierdo 1
Indice derecho 6
[14, 10, 0, 11, 9, 1, 20, 21, 24]
Valor del pivote 1
Indice izquierdo 1
Indice derecho 4
[1, 0, 10, 11, 9, 14, 20, 21, 24]
Valor del pivote 10
Indice izquierdo 3
Indice derecho 4
[0, 1, 10, 9, 11, 14, 20, 21, 24]
lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
```

```
#Importando bibliotecas
%pylab inline
import matplot.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
Populating the interactive namespace from numpy and matplotlib
```

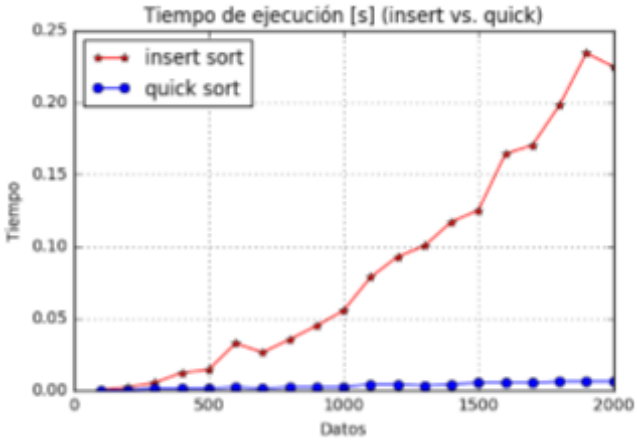
```python
#Cargando modulos
import random
from time import time

#Cargando las funciones guardadas en los archivo
from insertionSort import insertionSort_time
#Solo se necesita llamar a la funcion principal
from quickSort import quicksort_time

#Tamaños de la lista de numeros aleatorios a generar
datos = [ii*100 for ii in range(1,21)]

tiempo_is = [] #Lista para guardar el tiempo de ejecucion de insert sort
tiempo_qs = [] #Lista para guardar el tiempo de ejecucion de quick sort

for ii in datos:
    lista_is = random.sample(range(0, 10000000), ii)
    #Se hace una copia de la lista para que se ejecute el algoritmo con los mismos numeros
    lista_qs = lista_is.copy()

    t0 = time() #Se guarda el tiempo inicial
    quicksort_time(lista_qs)
    tiempo_qs.append(round(time()-t0, 6))

#Imprimiendo tiempos parciales de ejecucion
print("Tiempos parciales de ejecucion en INSERT SORT {} [s] \n".format(tiempo_is))
print("Tiempos parciales de ejecucion en QUICK SORT {} [s]".format(tiempo_qs))
```

```
Tiempos parciales de ejecución en INSERT SORT [0.0, 0.002005, 0.005013, 0.012001, 0.014037, 0.032504, 0.026106, 0.035
09, 0.044623, 0.055145, 0.078246, 0.092553, 0.100029, 0.116888, 0.124838, 0.16404, 0.170012, 0.197678, 0.234017, 0.22
473] [s]

Tiempos parciales de ejecución en QUICK SORT [0.0, 0.0, 0.001004, 0.001004, 0.001003, 0.002035, 0.000996, 0.002005,
0.002005, 0.002006, 0.004008, 0.004011, 0.003008, 0.004006, 0.005014, 0.005479, 0.005006, 0.005985, 0.006016, 0.00601
6] [s]
```

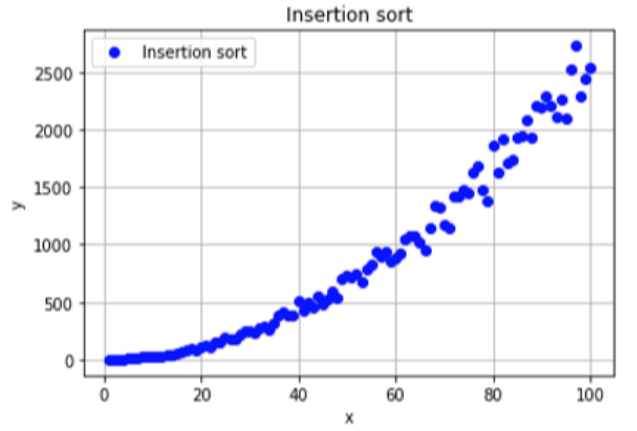
```python
#Imprimiendo tiempos totales de ejecucion
#Para calcular el tiempo total se aplica la funcion sum() a las listas de tiempo
print("Tiempos parciales de ejecucion en INSERT SORT {} [s]".format(sum(tiempo_is)))
print("Tiempos parciales de ejecucion en QUICK SORT {} [s]".format(sum(tiempo_qs)))
```

```
Tiempo total de ejecución en insert sort 1.729555 [s]
Tiempo total de ejecución en quick sort 0.06060699999999999 [s]
```

```python
#Generando la grafica
fig, ax = subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker"*",color="r")
ax.plot(datos, tiempo_qs, label="quick sort", marker"o",color="b")
ax.set_xlabel('Datos')
ax.set_ylabel('Tiempo')
ax.grid(True)
ax.legend(loc=2)

plt.title('Tiempo de ejecucion [s] (insert vs. quick)')
```




```
import matplot.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

times = 0

def insertionSort_graph(n_lista):
    global times
    for index in range(1,len(n_lista)):
        times += 1
        actual = n_lista[index]
        posicion = index
        while posicion>0 and n_lista[posicion-1]>actual:
            times += 1
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
    return n_lista
TAM = 101
eje_x = list(range(1,TAM,1))
eje_y = []
lista_variable = []

for num in eje_x:
    lista_variable = ramdom.sample(range(0, 1000), num)
    times = 0
    lista_variable = insertionSort_graph(lista_variable)
    eje_y.append(times)

fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(eje_x, eje_y, marker="o" ,color="b" , linestyle='None')

ax.set_xlabel('x')
ax.set_yabel('y')
ax.grid(True)
ax.legend(["Insertion sort"])

plt.title('Insertion sort')
plt.show()
```



Conclusión:

Al realizar los ejercicios me di cuenta de los diferentes usos y tipos de bibliotecas que utiliza el lenguaje Python. Siento que en mi caso se complicó un poco entender

el uso de estos pero con la ayuda de la práctica de nuevos ejercicios podre perfeccionar el uso de estos lenguajes.

Bibliografías y Cibergrafías:

- https://jupyter.org/try
- https://docs.python.org/3/library/itertools.html#
- https://docs.python.org/3/library/itertools.html#itertools.product
- https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
- https://docs.python.org/3.5/library/pickle.html
- Design and analysis of algorithms; Prabhakar Gupta y Manish Varshney; PHI Learning, 2012, segunda edición.
- Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein; The MIT Press; 2009, tercera edicion.
- Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edition.