	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado Martinez

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 13

No de Práctica(s). **11**

Integrante(s): **Mondragón Carrillo Luis Emir**

No. de Equipo de cómputo empleado:

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: **2020-2**

Fecha de entrega: **07/06/20**

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN:

Para esta práctica tomaremos en cuenta 6 diferentes estrategias las cuales son: Divide y vencerás, Fuerza bruta, Incremental, Greedy, top down y bottom down.

DESARROLLO:

1. Con la estrategia de “divide y vencerás” ordenamos número a partir de un pivote u posiciones, saliendo de un arreglo de números con el fin de que sea más fácil.

```
#Estrategia divide y venceras
'''
21 10 12 0 34 15
p   i          d

'''

def quicksort(lista):
    quicksort2(lista, 0, len(lista)-1)

def quicksort2(lista, inicio, fin):
    if inicio < fin:
        pivote = particion(lista, inicio, fin)
        quicksort2(lista, inicio, pivote-1)
        quicksort2(lista, pivote+1, fin)

def particion(lista, inicio, fin):
    pivote = lista[inicio]
    print("Valor el pivote {}",format(pivote))
    izquierda = inicio+1
    derecha = fin
    print("Indice izquierda {} y indice derecha {}".format(izquierda, derecha))
```

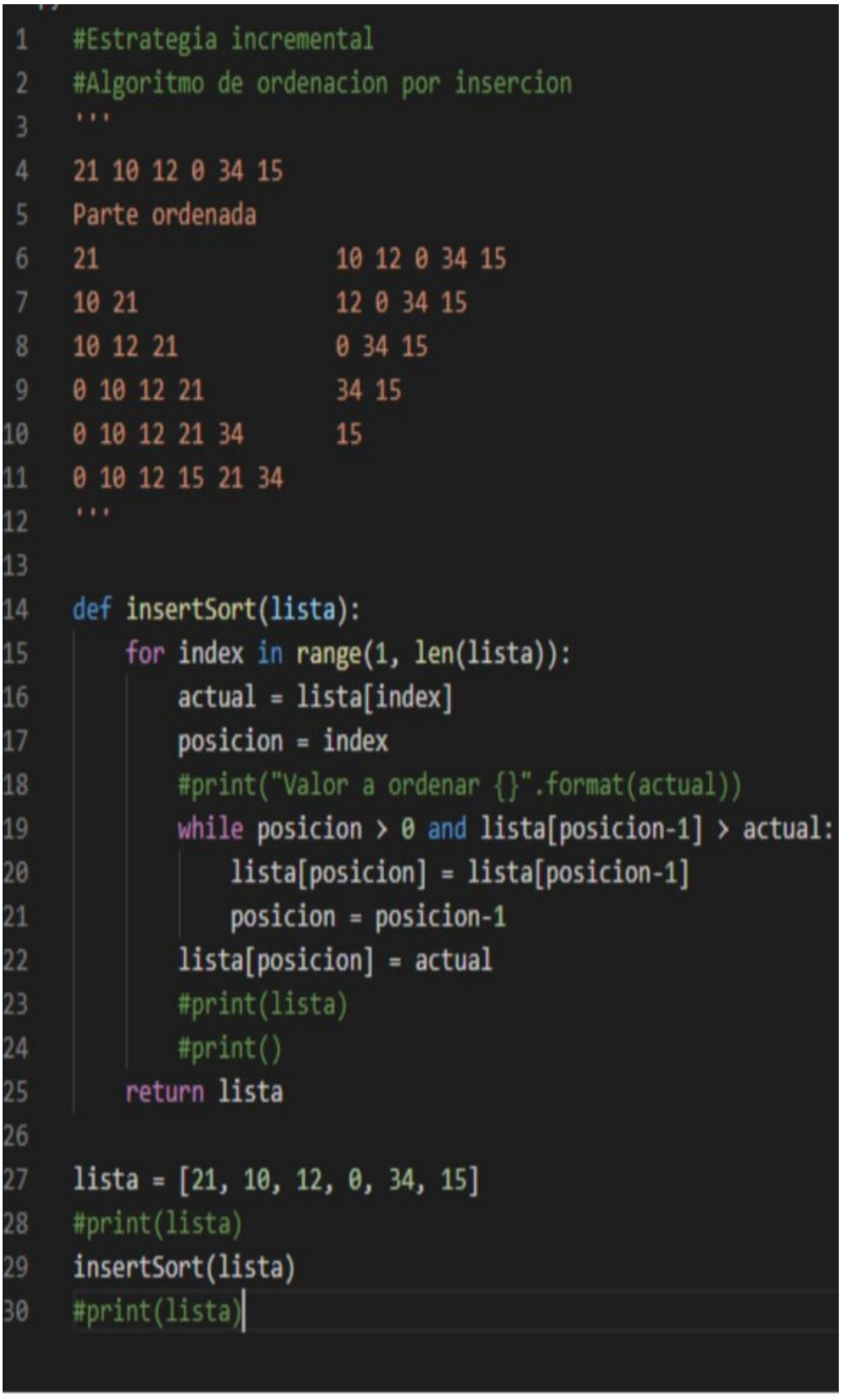
```

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while lista[derecha] >= izquierda and lista[derecha] >= pivote:
            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 derecha

lista = [21,10,0,11,9,24,20,14,1]
print(lista)
quicksort(lista)
print(lista)
```

2. La estrategia de fuerza bruta sirve generando combinaciones con el fin de hallar una contraseña, dependiendo de la potencia del equipo este tardará más o menos; lo primero es que abre un archivo de texto en el que se guardan todas las combinaciones y para obtener dichas combinaciones utilizamos la función del producto, contenida en la biblioteca itertools y por último un ciclo for que prueba cada una.

1.py

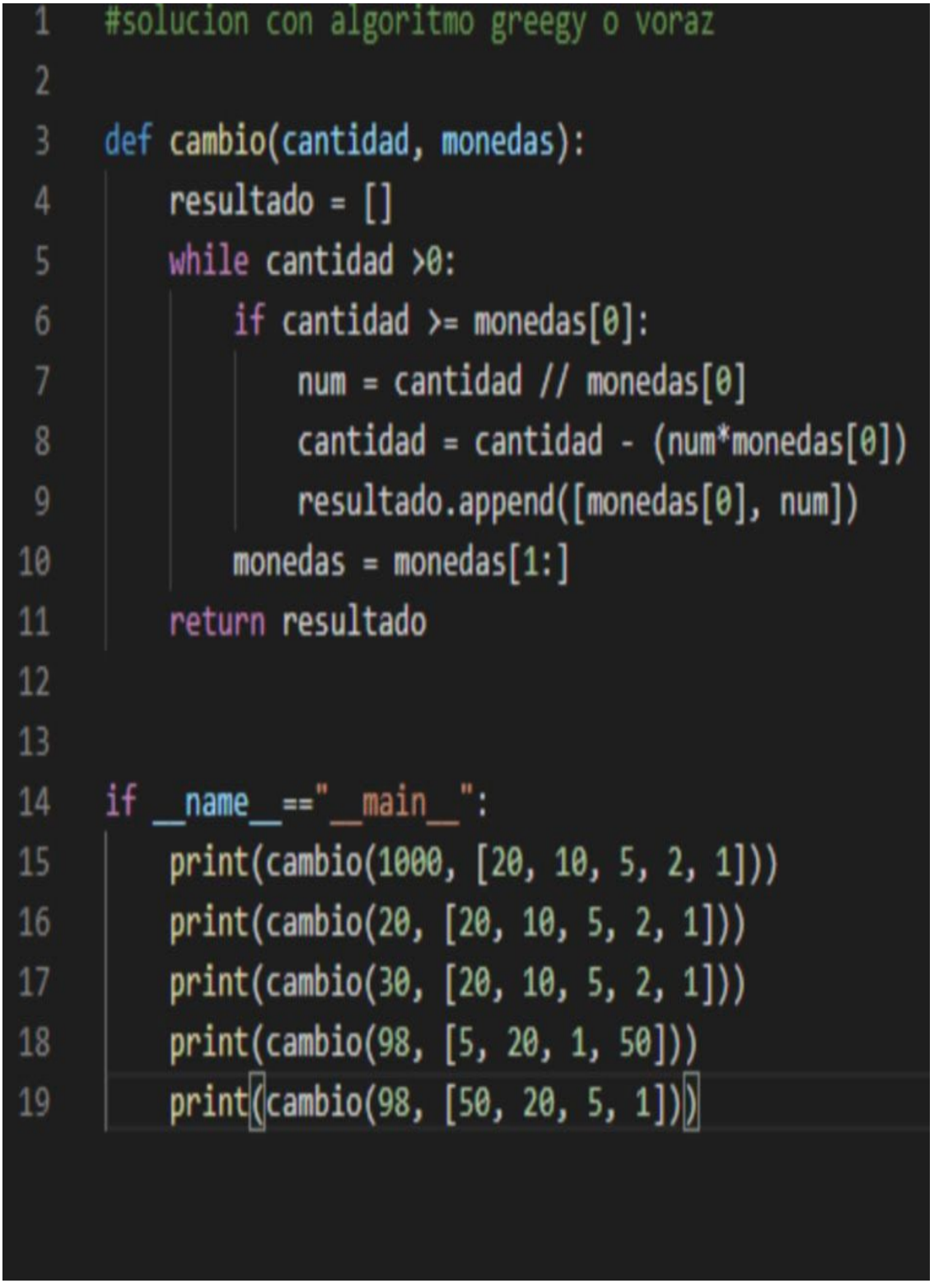
```python
#Estrategia de busqueda de fuerza bruta
#realiza una busqueda exhaustiva

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

caracteres = ascii_letters + digits

def buscar(con):
    #Abrir el archivo con las cadenas generadas
    archivo = open("combinaciones.txt", "w")

    if 3<= len(con) <= 4:
        for i in range(3, 5):
            for comb in product(caracteres, repeat = i):
                prueba = "".join(comb)
                archivo.write(prueba+"\n")
                if prueba == con:
                    print("La contraseña es {}".format(prueba))
                    break
        archivo.close()
    else:
        print("Ingresa una contraseña de longuitud 3 o 4")


if __name__=="__main__":
    con = input("Ingresa una contraseña\n")
    t0 = time()
    buscar(con)
    print("Tiempo de ejecucion {}".format(round(time()-t0,6)))
```

3. La estrategia incremental se usa y ordena una lista, parte de que el primer elemento esté ordenado, luego compara con el segundo y así consecutivamente.



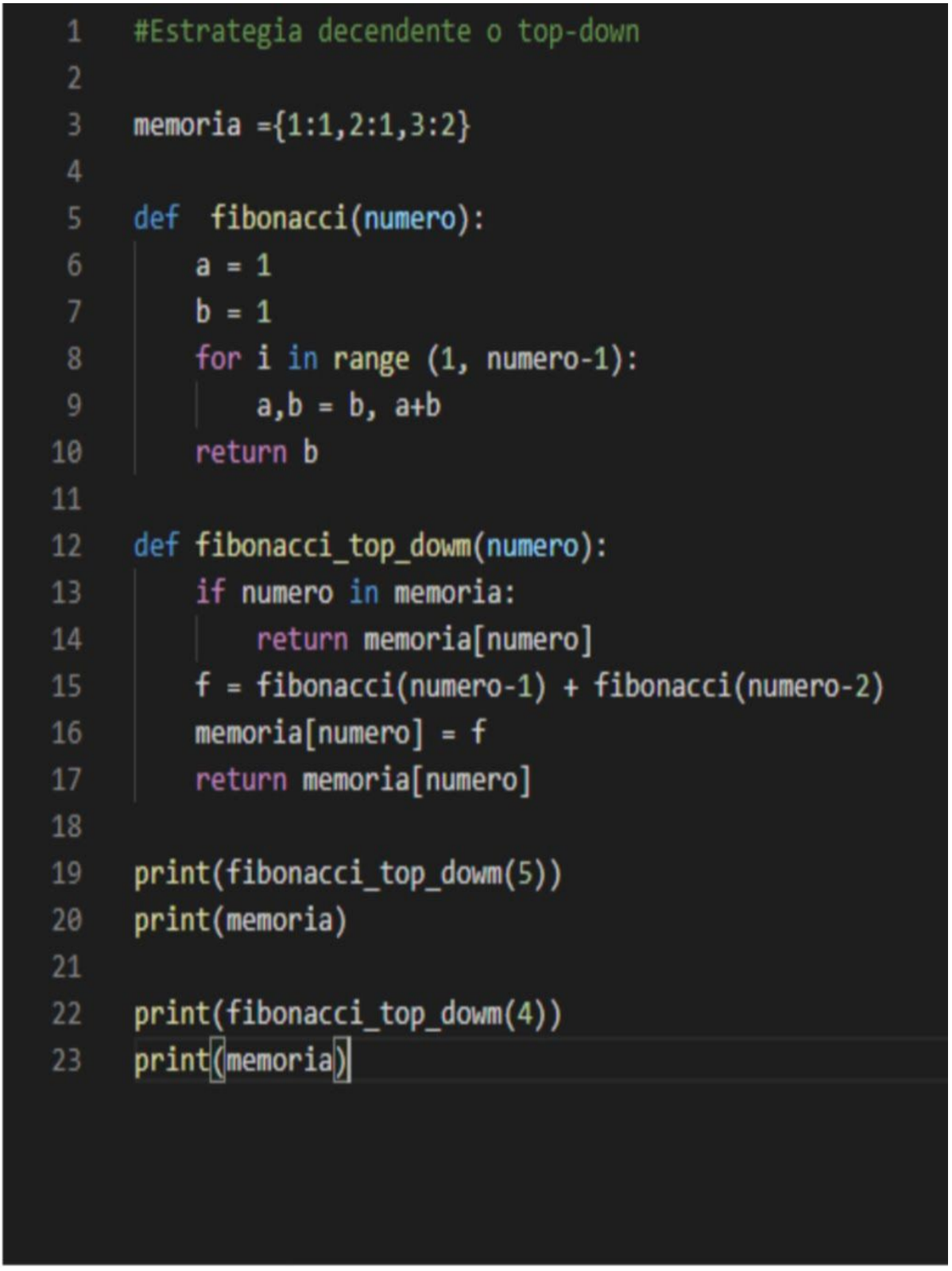
```
#Estrategia incremental
#Algoritmo de ordenacion por insercion
'''
21 10 12 0 34 15
Parte ordenada
21                  10 12 0 34 15
10 21               12 0 34 15
10 12 21            0 34 15
0 10 12 21          34 15
0 10 12 21 34       15
0 10 12 15 21 34
'''

def insertSort(lista):
    for index in range(1, len(lista)):
        actual = lista[index]
        posicion = index
        #print("Valor a ordenar {}".format(actual))
        while posicion > 0 and lista[posicion-1] > actual:
            lista[posicion] = lista[posicion-1]
            posicion = posicion-1
        lista[posicion] = actual
        #print(lista)
        #print()
    return lista

lista = [21, 10, 12, 0, 34, 15]
#print(lista)
insertSort(lista)
#print(lista)
```

4. En la estrategia greedy buscamos crear un programa que devolviera el cambio de monedas de acuerdo con la cantidad de dinero ingresado y el número de monedas que se tenía para dar cambio, para así devolver el menor número de monedas; se crea un algoritmo que selecciona las monedas desde el mayor hasta el límite y así usa las monedas del siguiente valor.



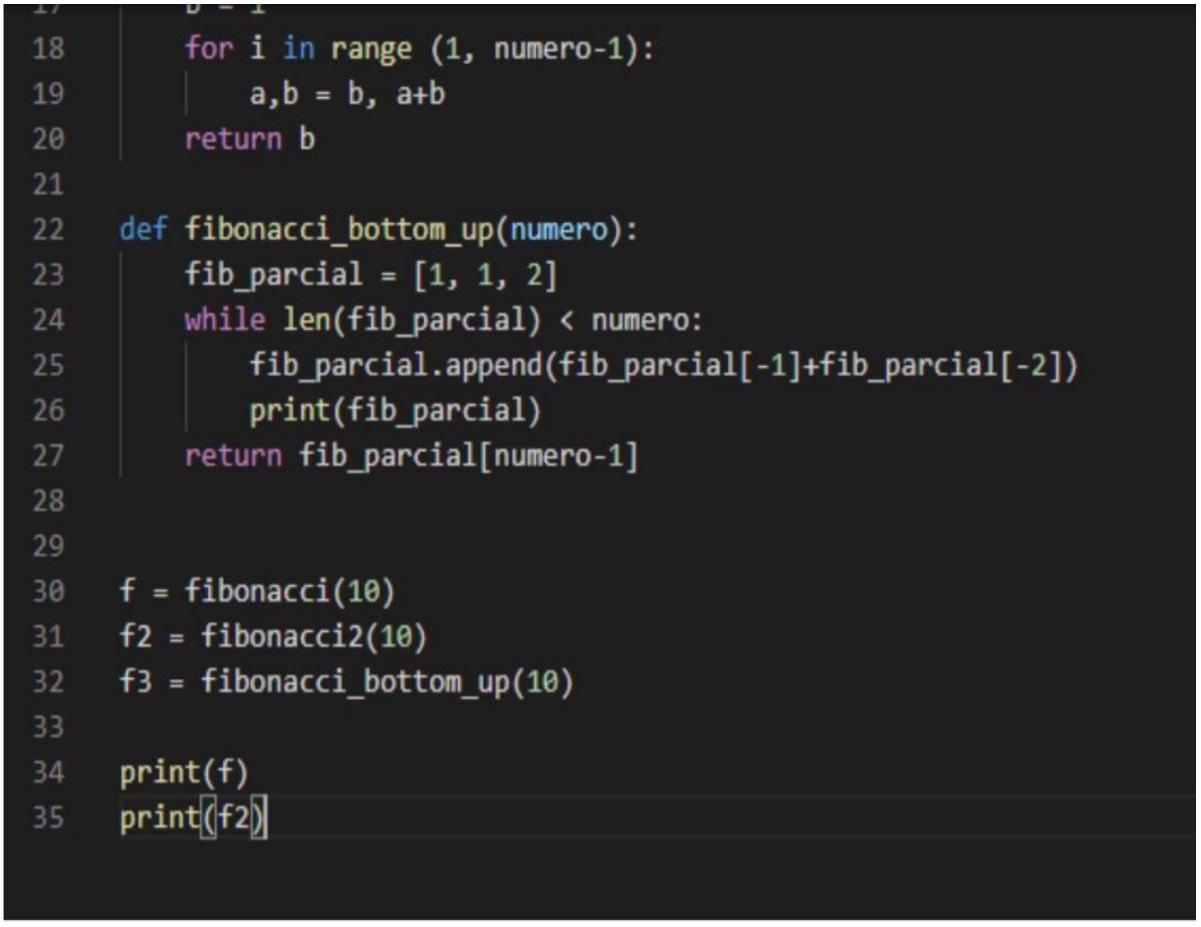
```python
#solucion con algoritmo greegy o voraz

def cambio(cantidad, monedas):
    resultado = []
    while cantidad >0:
        if cantidad >= monedas[0]:
            num = cantidad // monedas[0]
            cantidad = cantidad - (num*monedas[0])
            resultado.append([monedas[0], num])
        monedas = monedas[1:]
    return resultado


if __name__=="__main__":
    print(cambio(1000, [20, 10, 5, 2, 1]))
    print(cambio(20, [20, 10, 5, 2, 1]))
    print(cambio(30, [20, 10, 5, 2, 1]))
    print(cambio(98, [5, 20, 1, 50]))
    print(cambio(98, [50, 20, 5, 1]))
```

5. La estrategia top down resuelve lo mismo que button up solo que hace uso de un diccionario que guarda los resultados generados para cuando se requiera un término n, comprueba si existe dicho diccionario, si no sólo lo calculo y guarda.



```python
#Estrategia decendente o top-down

memoria ={1:1,2:1,3:2}

def  fibonacci(numero):
    a = 1
    b = 1
    for i in range (1, numero-1):
        a,b = b, a+b
    return b

def fibonacci_top_dowm(numero):
    if numero in memoria:
        return memoria[numero]
    f = fibonacci(numero-1) + fibonacci(numero-2)
    memoria[numero] = f
    return memoria[numero]

print(fibonacci_top_dowm(5))
print(memoria)

print(fibonacci_top_dowm(4))
print(memoria)
```

6. La estrategia buttom up su objetivo es obtener el término n de la sucesión de Fibonacci, las primeras dos funciones son la solución iteraria del problema mientras que la tercera es la que hace referencia a la estrategia buttom up en ella hay un arreglo de solocuionds parciales de los primeros números de manera que cuando se pida un cierto número de calculan los resultados desde los casos base, hasta llegar a n de abajo hacia arriba.

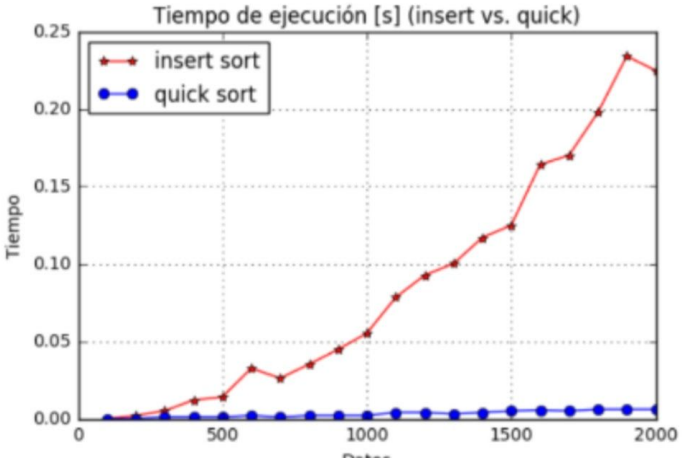


```
        for i in range (1, numero-1):
            a,b = b, a+b
        return b

def fibonacci_bottom_up(numero):
    fib_parcial = [1, 1, 2]
    while len(fib_parcial) < numero:
        fib_parcial.append(fib_parcial[-1]+fib_parcial[-2])
        print(fib_parcial)
    return fib_parcial[numero-1]


f = fibonacci(10)
f2 = fibonacci2(10)
f3 = fibonacci_bottom_up(10)

print(f)
print(f2)
```

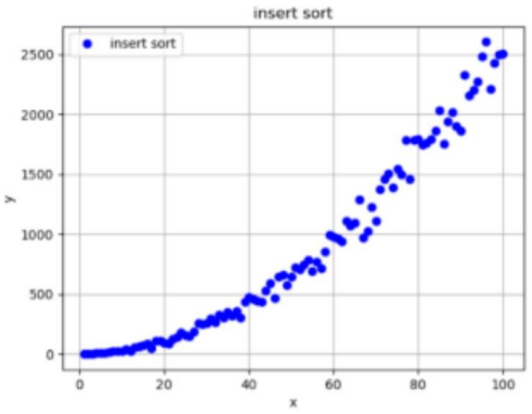
7. Los programas 7 y 8 son sobre gráficas y medición en tiempo de ejecución, aquí se comparan.

7.py

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random
from time import time
from ejercicio5 import insertSort
from ejercicio6 import quicksort

datos = [ii+100 for ii in range(1,21)]
tiempo_is = []
tiempo_qs = []
for ii in datos:
    lista_is = random.sample(range(0,10000000), ii)
    lista_qs =lista_is.copy()

    t0 = time()
    insertSort(lista_is)
    tiempo_is.append(round(time()-t0,6))

    t0 = time()
    quicksort(lista_qs)
    tiempo_qs.append(round(time()-t0,6))

print("Tiempos parciales de ejecucion en inserSort {}[s]".format(tiempo_is))
print("Tiempos parciales de ejecucion en quicksort {}[s]".format(tiempo_qs))

fig, ax = plt.subplots()
ax.plot(datos, tiempo_is, label="inserSort", marker="*", color ="r")
ax.plot(datos, tiempo_qs, label="quicksort", marker="o", color ="b")

ax.set_xlabel("Datos")
ax.set_ylabel("Tiempo")
ax.grid(True)
ax.legend(loc=2)

plt.title("Tiempos de ejecucion [s] inserSort vs quicksort")
plt.show()
```



8.
```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random

times = 0

def insertSort(lista):
    global times
    for i in range(1, len(lista)):
        times += 1
        actual = lista[i]
        posicion = i
        while posicion > 0 and lista[posicion-1] > actual:
            times += 1
            lista[posicion] = lista[posicion-1]
            posicion = posicion-1
        lista[posicion] = actual
    return lista

TAM = 101
eje_x= list(range(1, TAM,1))
eje_y = []

lista_variable = []

for num in eje_x:
    lista_variable = random.sample(range(0,1000), num)
    times = 0
    lista_variable = insertSort(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_ylabel('y')
ax.grid(True)
ax.legend(["insert sort"])

plt.title("Insert sort")
plt.show()
```



CONCLUSIÓN:

El lenguaje python resulta útil en esta práctica para conocer estrategias de solución de problemas que ya se utilizó en la práctica anterior como en la elaboración de gráficas además de conocer las propiedades del lenguaje Python.