|  |  |
| --- | --- |
|  | **Carátula para entrega de prácticas** |
| Facultad de Ingeniería | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | M.I. Marco Antonio Martinez Quintana |
| *Asignatura:* | estructuras de datos y algoritmos |
| *Grupo:* | 17 |
| *No de Práctica(s):* | 11 |
| *Integrante(s):* | Villanueva Corona Miguel Angel |
| *No. de Equipo de cómputo empleado:* | 36 |
| *No. de Lista o Brigada:* | 41 |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 21 de abril del 2020 |
| *Observaciones:* |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

***Objetivos***

El objetivo de esta guía es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos.

***Introducción***

En el momento que se nos presenta un problema tratamos de solucionarlo lo más pronto posible, estudiando sus causas para actuar correctamente, tomando en cuenta que un algoritmo es una serie de pasos bien definidos que ayudan a llegar a la solución de algún problema. Es así como, mediante las diferentes técnicas de diseño de algoritmos, construimos soluciones que satisfagan los requerimientos del problema.

***Desarrollo***

**Fuerza bruta**

**#Es un tipo método el cual es de forma exhaustiva lo que puede ser aveces lo mas eficaz a la hora de buscar alguna solución el único problema es que puede ser un proceso muy lento por que se va buscando de uno por uno**

from string import ascii\_letters , digits

from itertools import product

caracteres=ascii\_letters+digits

def buscador(con):

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("tu contraseña es {}".format(prueba))

archivo.close()

break

else:

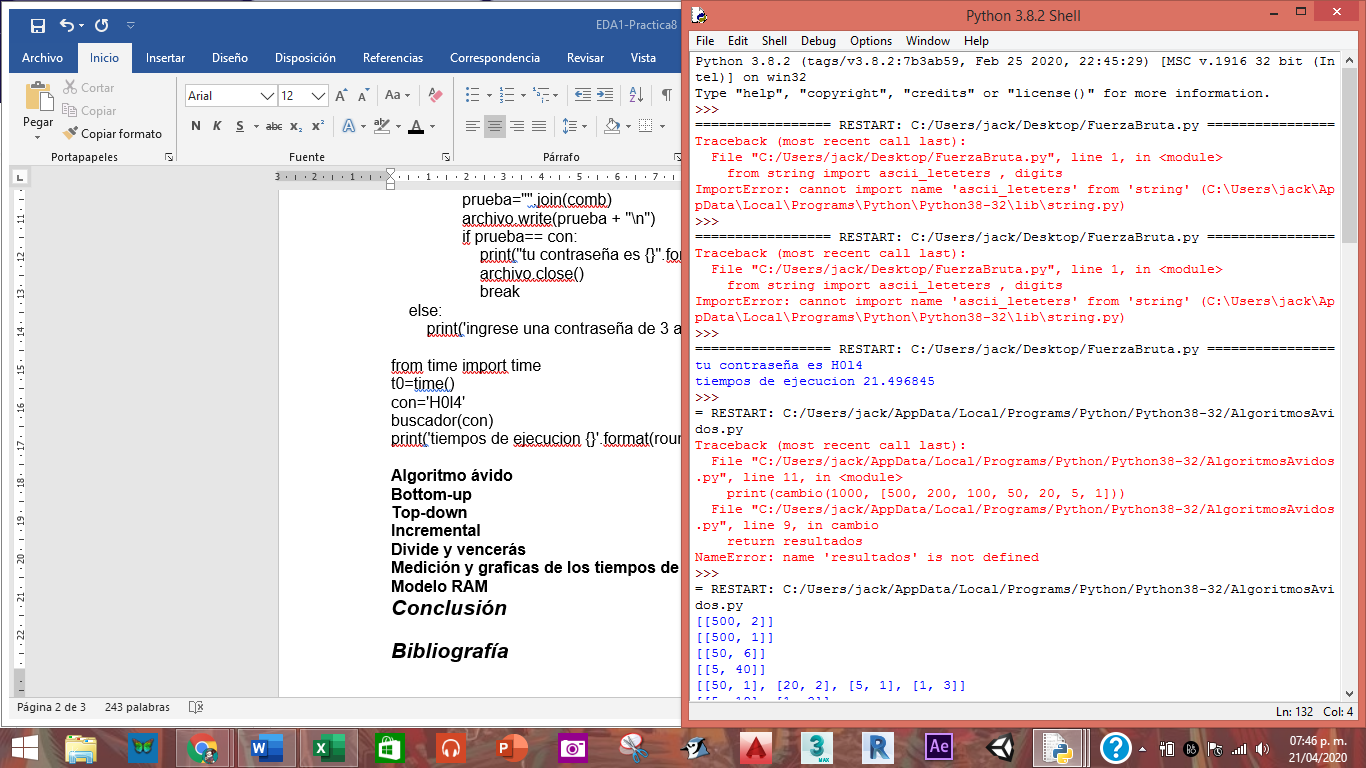
print('ingrese una contraseña de 3 a 4 caracteres')

from time import time

t0=time()

con='H0l4'

buscador(con)

print('tiempos de ejecucion {}'.format(round(time()-t0,6)))

**Algoritmo ávido**

**#Este tipo de algoritmo es lo contrario al de fuerza bruta este proceso es muy rápido pero el único defecto es que no siempre se encuentra la solución mas eficaz**

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:]

return resultado

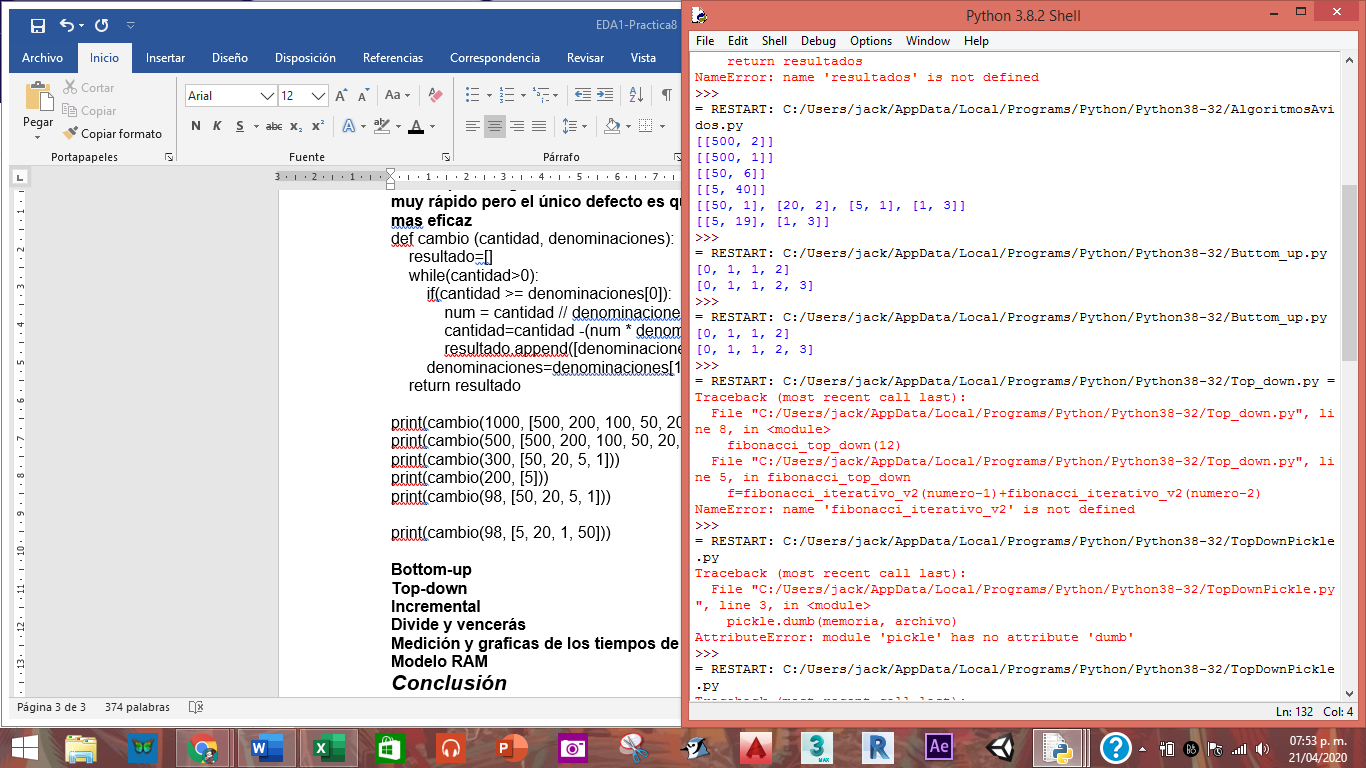
print(cambio(1000, [500, 200, 100, 50, 20, 5, 1]))

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]))

**Bottom-up**

**#Esta forma se basa en resolver un problema a base de subproblemas ya resueltos con anticipación y la suma de todas estas resoluciones ayudan a la resolución del problema principal.**

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\_v1(6)

def fibonacci\_buttom\_up(numero):

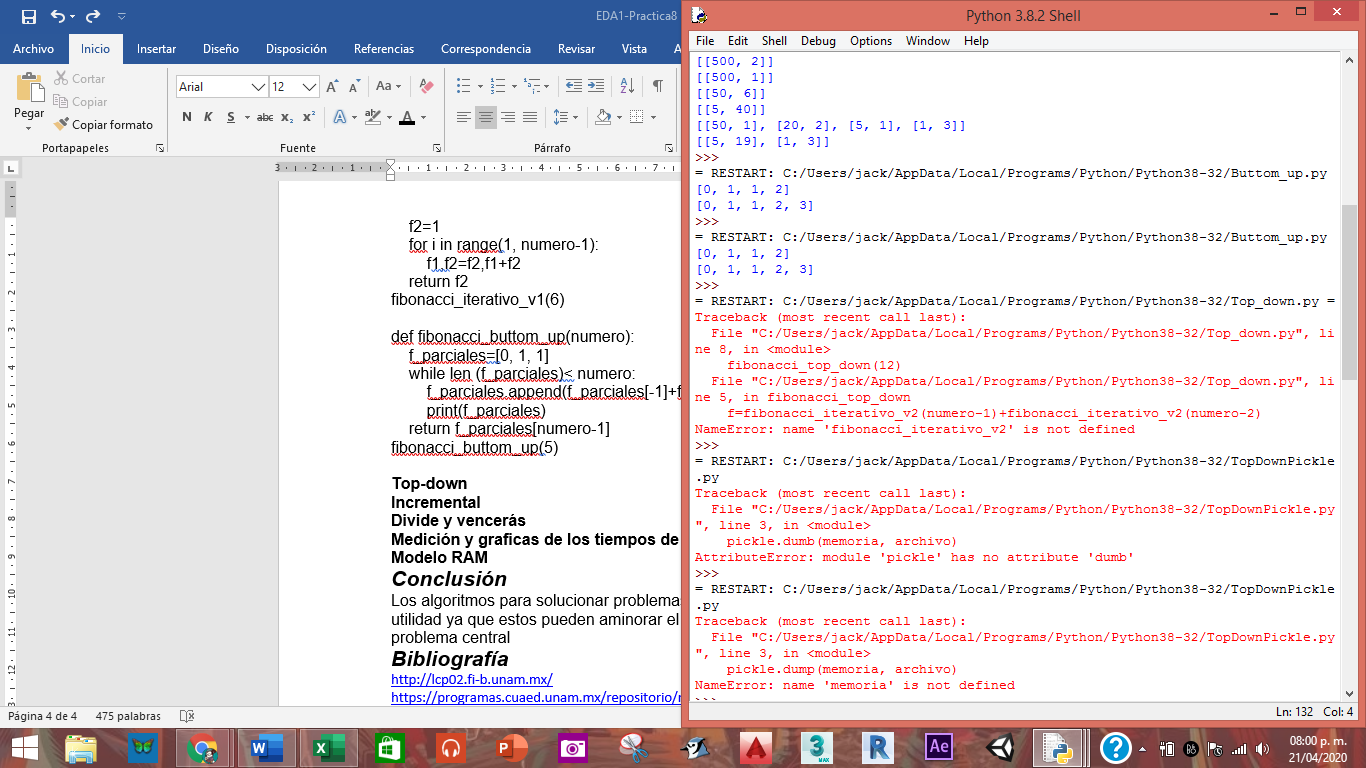
f\_parciales=[0, 1, 1]

while len (f\_parciales)< numero:

f\_parciales.append(f\_parciales[-1]+f\_parciales[-2])

print(f\_parciales)

return f\_parciales[numero-1]

fibonacci\_buttom\_up(5)

**Top-down**

**#Este método es similar a Button-up la una diferencia es que este hace cálculos de abajo hacia arriba**

memoria = [1:0, 2:1, 3:1]

def fibonacci\_top\_down(numero):

if numero in memoria:

return memoria[numero]

f=fibonacci\_iterativo\_v2(numero-1)+fibonacci\_iterativo\_v2(numero-2)

memoria[numero]=f

return memoria[numero]

fibonacci\_top\_down(12)

memoria

fibonacci\_top\_down(8)

memoria

import pickle

archivo=open('memoria.py','wb')

pickle.dump(memoria, archivo)

archivo.close()

archivo=open('memoria.p','rb')

memoria\_de\_archivo=pickle.load(archivo)

archivo.close()

memoria

memoria\_de\_archivo

**Incremental**

**#Consiste en solucionar un problema implementar y probar estrategias para agregar información paulatinamente**

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)

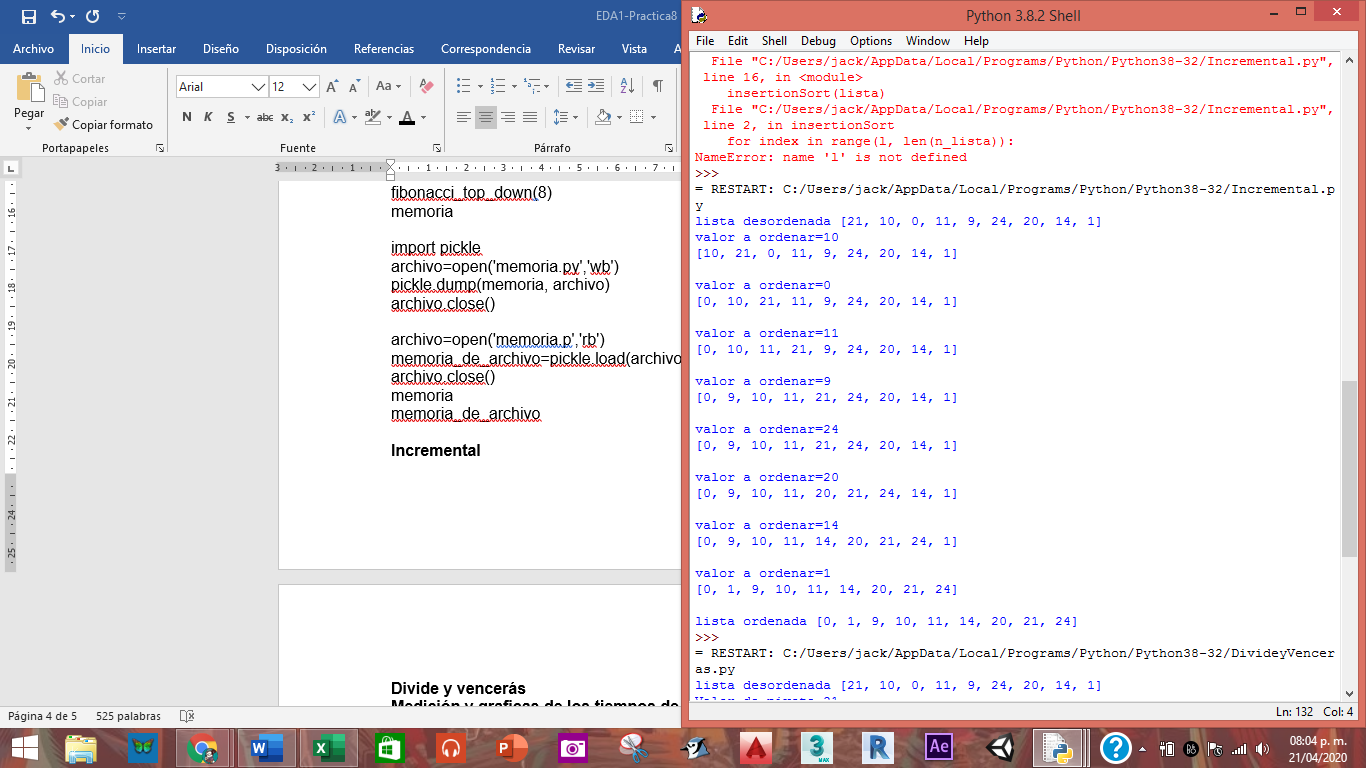
print()

return n\_lista

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

print('lista desordenada {}'.format(lista))

insertionSort(lista)

print('lista ordenada {}'.format(lista))

**Divide y vencerás**

**#Como dice este método se basa en separar el problema en subproblemas para resolverlos por separados para que al final juntarlos todos y asi darle solución al problema central**

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):

pivote=lista[inicio]

print('Valor de pivote {}'.format(pivote))

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 derecha

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

print('lista desordenada {}'.format(lista))

quicksort(lista)

print('lista ordenada {}'.format(lista))

**Medición y graficas de los tiempos de ejecución**

%pylab inline

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import random

from time import time

from insertionSort import insertionSort\_time

from quickSort import quicksort\_time

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()

insertionSort\_time(lista\_is)

tiempo\_is.append(round(time()-t0, 6))

t0=time()

quicksort\_time(lista\_qs)

tiempo\_qs.append(round(time()-t0, 6))

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))

print('Tiempos total de ejecucion en insert sort {} [s]'.format(sum(tiempo\_is)))

print('Tiempos total de ejecucion en quick sort {} [s]'.format(sum(tiempo\_qs)))

fig, ax=subplots()

ax.plot(datos, tiempo\_is, label='insert sort', marker='\*',color='r')

ax.plot(datos, tiempo\_qs, label='quick sort', marker='°',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()

**Modelo RAM**

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

import matplotlib.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=random.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\_ylabel('y')

ax.grid(True)

ax.legend(['insertion sort'])

plt.title('insertion sort')

plt.show()

***Conclusión***

Los algoritmos para solucionar problemas son varios que son de muy buena utilidad ya que estos pueden aminorar el tiempo en el cual se resuelve un problema central

***Bibliografía***

<http://lcp02.fi-b.unam.mx/>

<https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/1196/mod_resource/content/1/contenido/index.html>