Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

PROFESOR:	Adrian ulises mercado
ASIGNATURA:	Estructura de datos y Algoritmos I
GRUPO:	13
NO DE PRÁCTICA(S):	Practica 11
INTEGRANTE(S):	Jose Abraham Hernandez Vargas
NO. DE EQUIPO DE CÓMPUTO EMPLEADO:	
NO. DE LISTA O BRIGADA:	Brigada 5
SEMESTRE:	2020-2
FECHA DE ENTREGA:	07/06/2020
OBSERVACIONES:	

CALIFICACIÓN: _____

INTRODUCCIÓN

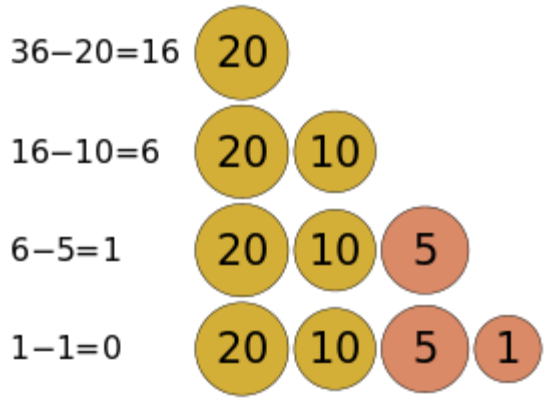
Esta práctica tiene como objetivo la comprensión e implementación de las estrategias para la construcción de algoritmos, a continuación, una breve explicación de algunas estrategias.

Fuerza bruta

Esta estrategia consiste en hacer una búsqueda exhaustiva de todas las posibilidades que lleven a la solución del problema. Es el algoritmo mas simple que existe, consiste en probar todas las posibles soluciones del código, hablando de la lógica que lleva esta estrategia se situa en una a posición y compara carácter a carácter hasta encontrar un fallo o una solución final.

Algoritmos ávidos (greedy)

Esta estrategia no es tan simple y se diferencia de la fuerza bruta por tomar una serie de decisiones en orden especifico, una vez ejecutada esa decisión, ya no se vuelve a considerar. Ejemplo:



Bottom-up(programación dinámica)

En este tipo de algoritmos antes de resolver un problema completamente se busca que si se pueden resolver problemas iguales o similares, pero de menor escala se resuelvan y después estos se usen para resolver problemas mayores. Ejemplo más claro es del algoritmo de Fibonacci

Top-down

Esta estrategia de programación consiste en que el problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memorización y recursión.

Algoritmos Incrementar

Un algoritmo incrementar es parcialmente dinámico donde se va actualizando cada que se hace una inserción, el ejemplo más claro de este tipo de algoritmos seria la función insertsort

Divide y vencerás

Como lo dice el nombre de este tipo de algoritmos este sirve para resolver un problema difícil dividiéndolo en partes más simples tantas como sea necesario hasta que la solución se hace obvia. Esta estrategia está basada en el método de la solución recursiva.

DESARROLLO

En esta practica veremos conceptos como el enfoque grredy, el ordenamiento por mezcla Quicksort y meregesort asi mismo veremos los algoritmos de fuerza bruta que no son muy eficientes pero siempre encontraran la solución tarden lo que tarden.

Ejercicio1.py

ejercicio1.py > ...

```python
#Estrategia de fuerza bruta
#Realiza una busqueda exaustiva
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 contrasena es {}",format(prueba))
                    archivo.close()
                    break

    else:
        print("Ingrese una contrasena de longitud 3 o 4")

if __name__=="__main__":
    con = input("ingresa una contrasena\n")
    t0 = time()
    buscar(con)
    print("tiempo de ejecuccion {}".format(round(time()-t0,6)))
```

Este algoritmo esta basado en la estrategia de “Fuerza bruta”, este algoritmo sirve para encontrar una contrasena de 4 caracteres y empieza buscando desde la solucion ms simple que seria “aaaa” y asi va probando contrasenas hasta que encuentra la solucion, este tipo de algoritmos son de los mas ineficientes pero funciona asi que seria de las ultimas opciones.

Ejecucion

```
4766695   tack
4766696   tacl
4766697   tacm
4766698   tacn
4766699   taco
4766700

OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS 5

ingresa una contrasena
taco
la contrasena es {} taco
tiempo de ejecuccion 8.096416
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

Se puede observar que el tiempo de ejecución fue de ocho segundos y que genero mas de 4 millones de posibles resultados

Ejercicio2.py

```python
# Algoritmo greedy
def cambio(cantidad, monedas):
    resulatdo = []
    while cantidad > 0:
        if cantidad >= monedas[0]:
            num = cantidad//monedas[0]
            cantidad = cantidad - (num*monedas[0])
            resulatdo.append([monedas[0], num])
        monedas = monedas[1:]
    return resulatdo

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

```
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11> & C:/Users/dan
a 11/ejercicio2.py"
[[20, 50]]
[[20, 1]]
[[20, 1], [10, 1]]
[[50, 1], [20, 2], [5, 1], [1, 3]]
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

Es un algortimo de tipo greedy

Ejercicio3.py

```python
    a = 1
    b = 1
    c =0
    for i in range(1, numero-1):
        c=a +b
        a = b
        b = c
    return c

def fibonacci2(numero):
    a = 1
    b = 1
    for i in range(1, numero-1):
        a, b= b,a+b
    return b

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

f = fibonacci(4)#El fibonacci de 4=3
print(f)
```

```
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11> & C:/Users/dany/
a 11/ejercicio3.py"
3
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

Es un algoritmo de tipo bottom-up, aquí busca calcular el Fibonacci de un numero, consiste en calcular el Fibonacci de números anteriores que serian mas fáciles de calcular y sumarlos para dar con la solución del problema.

Ejercicio4.py

```python
#estrategia descendente o top-down
memoria= {1:1, 2:1, 3:1}

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

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

print(fibonacci_top_down(5))
print(memoria)

print(fibonacci_top_down(4))
print(memoria)
```

```
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11> & C:/
a 11/ejercico 4.py"
5
{1: 1, 2: 1, 3: 1, 5: 5}
3
{1: 1, 2: 1, 3: 1, 5: 5, 4: 3}
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

El ejercicio 4 usa estrategia top-down, que ocupa recursividad y memoria. Por eso se tiene la variable memoria y la función Fibonacci para calcular lo que hay en memoria y después calcular Fibonacci para grandes como el caso de 5 y 4.

Ejercicio5.py

Este algoritmo es la función insertsort, y como lo había mencionado antes es un algoritmo de tipo incrementar, esta lo que hace es acomodar una serie de números de mayor a menor, esta funciona acomodando numero en una parte ordenada y en la que no, compara si el número que va a acomodar es menor al mayor que está en la parte acomodad y después lo actualiza con todos los que están acomodados.

```python
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)
```

```
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11> & C:/Users
a 11/ejercicio5.py"
[21, 10, 12, 0, 34, 15]
[0, 10, 12, 15, 21, 34]
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

Ejercicio6.py

Algoritmo divide y venceras, este algoritmo tambien busca ordenar una serie numeros pero esta vez vamos a ir diviendo la lista en partes para hasta que sea lo mas facil de ordenar y al final junta todos los problemas resueltos para llegar al ordenieto total.

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS 3

Valor del pivote 15
indice izquierdo 1 y indice derecho 3
[15, 10, 12, 0, 21, 34]
Valor del pivote 0
indice izquierdo 1 y indice derecho 2
[0, 10, 12, 15, 21, 34]
Valor del pivote 10
indice izquierdo 2 y indice derecho 2
[0, 10, 12, 15, 21, 34]
[0, 10, 12, 15, 21, 34]
PS C:\Users\dany\Desktop\Clases en linea EDA\Practica 11>
```

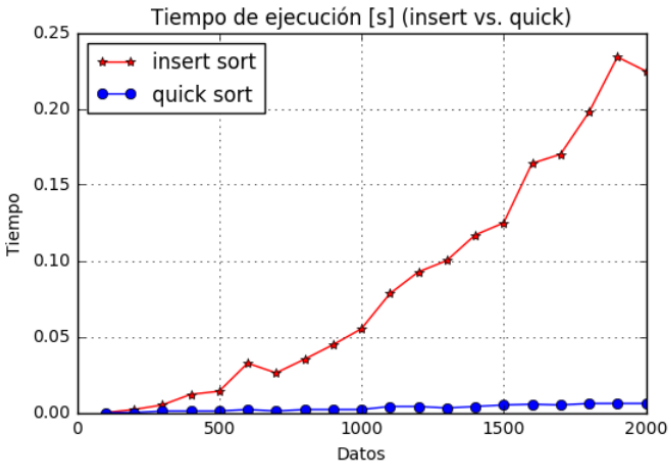
ejercicio6.py > particion

```python
#Divide y venceras
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 del pivote {}".format(pivote))
    izquierda = inicio + 1
    derecha = fin
    print("indice izquierdo {} y indice derecho {}" .format(izquierda, derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while 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, 12, 0, 34, 15]
print(lista)
quickSort(lista)
print(lista)
```

Ejercicio7.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 quicksor

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

print("tiempos parciales de ejecucion en insert sort {} [s]".format(tiempo_is))
print("tiempos parciales de ejecucion en quick sort {} [s]".format(tiempo_qs))

ax= plt.subplot(111)
ax.plot(datos, tiempo_is, label="insert sort",marker="*",color="r")
ax.plot(datos, tiempo_is, label="quick sort",marker="o",color="b")
```

```
ax.set_xlabel("datos")
ax.set_ylabel("tiempos")
ax.grid(True)
ax.legend(loc=2)

plt.title("tiempos de ejecucion [s] insert sort vs quick sort")
plt.show()
```

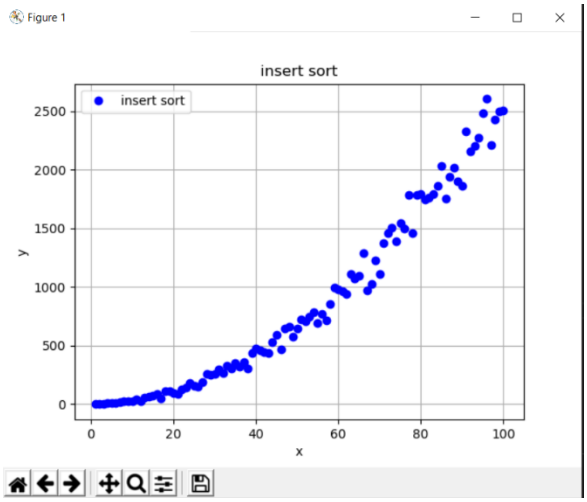


Para este ejercicio tenemos a las funciones insert sort y quicksort combatiendo ´para saber cual es la mas eficiente, utilizando la graficacion de datos con ax en Python y la biblioteca matplotlib podemos ver que el más eficiente en cuanto a tiempo es quicksort, pero en cuanto a datos tienen la misma cantidad, seria cuestión de ponerlo realmente a prueba con un problema a solucionar y averiguar si aumentamos mas datos, puede que quick sort aumente tiempos.

Ejercicio8.py

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

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()

#graficar quicksort con el modelo RAM
```



Para el caso del ejericio 8 tenemos la prueba y testeo de la función insert sort de la biblioteca de matplotlib y otras en Python, utilizando la graficacion en Python podemos ver que varía mucho la línea exponencial de insert sort, pocos datos puede traducirse en más tiempo de realización

COMENTARIOS

Me pareció una práctica muy completa, los ejercicios fueron muy entendibles para poder explicar los tipos de algoritmos.

CONCLUSION

Para cada problema puede haber diferentes soluciones, y va a haber diferentes estrategias que nos van a convenir mas

BIBLIOGRAFIA

https://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s#:~:text=En%20la%20cultura%20popular%2C%20divide,construye%20con%20las%20soluciones%20encontradas.
http://lcp02.fi-b.unam.mx/#