



## Caratula para entrega de Prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

Profesor: \_\_\_\_\_ Ing. Marco Antonio Martinez Quintana \_\_\_\_\_

Asignatura: \_\_ Estructura de Datos y Algoritmos I (1227) \_\_

Grupo: \_\_\_\_\_ 17 \_\_\_\_\_

No. de Práctica(s): \_\_\_\_\_ 11° \_\_\_\_\_

Integrante(s): \_\_\_\_\_ Avila Laguna Ricardo \_\_\_\_\_

No. de Equipo de  
cómputo empleado: \_\_\_\_\_ 10 \_\_\_\_\_

No. Lista o Brigada: \_\_\_\_\_ 6 \_\_\_\_\_

Semestre: \_\_\_\_\_ 2° \_\_\_\_\_

Fecha de entrega: \_\_\_\_\_ Abril del 2020 \_\_\_\_\_

Observaciones: \_\_\_\_\_

CALIFICACIÓN: \_\_\_\_\_

## 1° 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.

## 2° Introducción

Conceptos a revisar en Python:

- ❖ Escribir y leer en archivos.
- ❖ Graficar funciones usando la biblioteca Matplotlib.
- ❖ Generar listas de números aleatorios.
- ❖ Medir y graficar tiempos de ejecución.

## 3° Desarrollo y Resultados

**Actividades:**

- Revisar el concepto y un ejemplo de diversas estrategias para construir algoritmos (fuerza bruta, algoritmo ávido, bottom-up, top-down, divide y vencerás, etc).

**Código:**

```
*Practica11.py - C:\Users\Marbella\Documents\ProyectosPython\Practica11.py (3.8.2)*
File Edit Format Run Options Window Help

#Fuerza bruta
from string import ascii_letters,digits
from itertools import product
from time import time
#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 conctenar los caracteres regresado por la
                #Como join necesita una cadena inicial para hacer la concatenacion
                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('Ingrese una contraseña que contenga de 3 a 4 caracteres')
    t0 = time()
    con = 'H014'
    buscador(con)
    print("Tiempo de ejecucion {}".format(round(time()-t0,6)))
```

En el primer programa llamado “Fuerza Bruta” intenta realizar varias combinaciones entre caracteres y enteros para descifrar una contraseña, las mismas se van a ir guardando en un archivo de texto e imprimirá el tiempo que se tarde para descifrar la combinaciones posibles..

```
Practica11.py - C:\Users\Marbella\Documents\ProjetsPython\Practica11.py (3.8.2)
File Edit Format Run Options Window Help

#Bottom-up (programacion dinamica)
#Fibonacci
def fibonacci(n):
    f1 = 0
    f2 = 1
    tmp = 0
    for i in range(1,n-1):#For <constante> in range(<numero inicial(1)>,<numero final>
        tmp = f1+f2
        f1 = f2
        f2 = tmp
        print(f2)
    return f2
num = int(input('Ingresa un numero: '))
print("El numero de fibonacci en la pocicion {} es {}".format(num, fibonacci(num)))
#Asignacion paralela
def fibonacci2(n):
    f1 = 0
    f2 = 1
    tmp = 0
    for i in range(1,n-1):
        f1,f2=f2,f1+f2 #La , funciona como pasar y el = es como tambien
        #Ejemplo f1+f2 se pasa(,) a f2 tambien(=) f2 se pasa(,) a f1
        #print(f2)
    return f2
num2 = int(input('Ingresa un numero: '))
print("El numero de fibonacci en la pocicion {} es {}".format(num2, fibonacci2(num2)))
#estrategia bottom-up
#Len() devuelve la longitud de una cadena o loss elementos de una lista y se puede
def fibonacci_bottom_up(numero):
    f_parciales = [0,1,1] #Esto es una lista y para nombrar los datos de izquierda
    #derecha se inicia desde 0 hasta el final y de DERECHA a IZQUIERDA se inicia
    #-1,-2,-3 hasta llegar al inicio
    while len(f_parciales)< numero:
        print('-1 es {} y -2 es {}'.format(f_parciales[-1],f_parciales[-2]))
        f_parciales.append(f_parciales[-1] + f_parciales[-2])
        print(f_parciales)
    return f_parciales[numero-1]
numero = int(input('Ingrese un numero para fibnachi:'))
fibonacci_bottom_up(numero)
print('\n')

#Top-down
```

Ln: 251 Col: 0

Damos inicio a la programación DINÁMICA, es una de las herramientas más utilizadas en la programación, para darnos unos ejemplos generamos una función que nos diga la cantidad en una posición en la serie de fibonacci, y después utilizamos esa función para anidarla en otra función que realizará toda la serie hat el número de elementos que queramos .

```
Practica11.py - C:\Users\Marbella\Documents\ProjetsPython\Practica11.py (3.8.2)
File Edit Format Run Options Window Help

#Acomodo de datos
#InsertionSort (BURBUJA)
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))
print('\n')

#Quick Sort (Divide y vencerás)
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 derecha >= izquierda and lista[derecha] >= pivote:
            derecha = derecha - 1
        if izquierda < derecha:
            lista[izquierda], lista[derecha] = lista[derecha], lista[izquierda]
        else:
            bandera = True
    lista[inicio], lista[derecha] = lista[derecha], lista[inicio]
    return derecha
```

Tenemos una dos función que nos ayudará con el acomodo de datos en orden de mayor a menor, después con estas funciones declararemos varias listas para ordenarla con las funciones y calcular el tiempo en que requiere cada cálculo, para seguir con una gráfica comparativa entre tiempo de ejecución.



```
Practica11.py - C:\Users\Marbella\Documents\ProjetsPython\Practica11.py (3.8.2)
File Edit Format Run Options Window Help

plt.ylabel('Tiempo')

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

#Modelo RAM
times = 0
def insertionSort_graph(n_lista):
    global times
    for index in range(1, len(n_lista)):
        times += 1 #i++ o i+=1 o i=i+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_variables = insertionSort_graph(lista_variable)
    eje_y.append(times)

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

plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend(["Insertion sort"])

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

Ln: 251 Col: 0

Para otro ejemplo simularemos una RAM donde pedimos a una función calcule una serie de procedimientos y los compararemos en una gráfica para que sea de una mejor manera visual y permite entender mejor los datos.



```
#Se carga la biblioteca
import pickle
#Guardar variables en un archivo binario
#Generalmente se pone la extension .p o .pickle como estandar
archivo = open('memoria.p', 'wb')
pickle.dump(memoria, archivo)
archivo.close()

#Leer una variable de un archvo binario
archivo = open('memoria.p', 'rb')
memoria_de_archivo = pickle.load(archivo)
archivo.close()

print('La memoria es: {}'.format(memoria))
print('La memoria del archivo es: {}'.format(memoria_de_archivo))
print('\n')
```

Por último aprendimos a pasar una lista generada a un archivo .pickle de forma binaria para después leerla en el mismo programa y comparamos los resultados de las 2 listas para concluir que se guardaron y leyeron con éxito.

## 4° Conclusiones

Avila Laguna Ricardo :

Los objetivos se cumplieron porque aprendimos y comprendimos cómo utilizar la programación dinámica, junto con programas para manejo de datos para graficarlos e incluso guardarlos en un archivo, el único problema es que algunas líneas de código están erróneas para lo que pude darme cuenta y corregirlas, espero que pronto puedan corregir los errores.

## Bibliografía

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