



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrián Ulises Mercado Martínez

Asignatura: Estructura de Datos y Algoritmos I.

Grupo: 13

No de Práctica(s): 11

Integrante(s): Nicolás López Daniela

*No. de Equipo de
cómputo empleado:* -

No. de Lista o Brigada: 36

Semestre: 2020-2

Fecha de entrega: 07-06-2020

Observaciones:

CALIFICACIÓN: _____

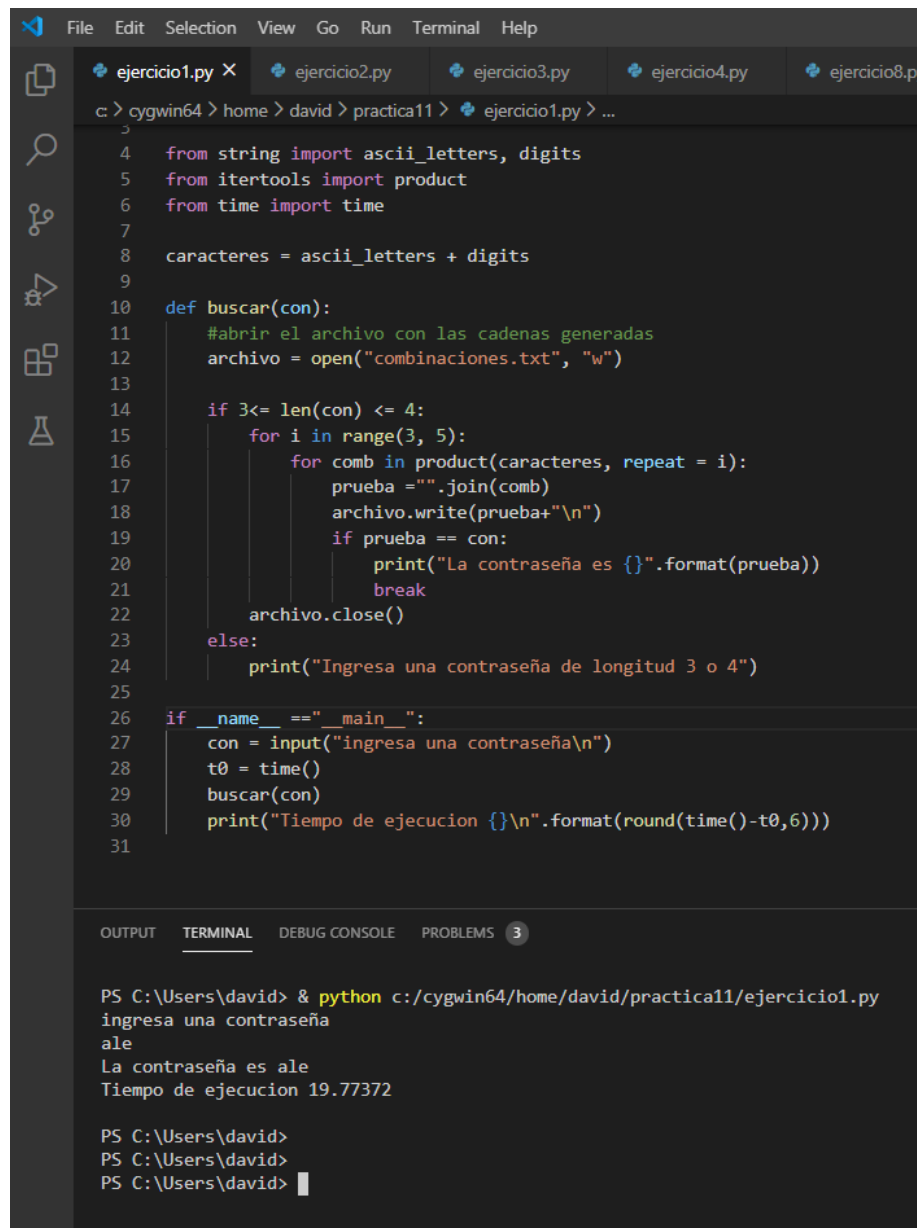
❖ INTRODUCCION.

En esta práctica realizamos 8 programas en los cuales creamos por medio de diferentes librerías y funciones algoritmos de distintos tipos, además de que por medio de la creación de graficas pudimos ver la eficiencia de algunos.

❖ DESARROLLO (CON EJERCICIOS)

▪ PROGRAMA 1.

En este primer programa realizamos un algoritmo de fuerza bruta, el programa se encarga de encontrar una contraseña de entre 3 o 4 caracteres, esto se realiza creando todas las combinaciones posibles que existan con esa cantidad de elementos, en este ejemplo colocamos las combinaciones en un archivo de texto, estos algoritmos no son muy eficientes ya que se realizan búsquedas exhaustivas del problema y el tiempo de ejecución es bastante elevado.



```
File Edit Selection View Go Run Terminal Help
ejercicio1.py X ejercicio2.py ejercicio3.py ejercicio4.py ejercicio8.p
c > cygwin64 > home > david > practica11 > ejercicio1.py > ...
4 from string import ascii_letters, digits
5 from itertools import product
6 from time import time
7
8 caracteres = ascii_letters + digits
9
10 def buscar(con):
11     #abrir el archivo con las cadenas generadas
12     archivo = open("combinaciones.txt", "w")
13
14     if 3<= len(con) <= 4:
15         for i in range(3, 5):
16             for comb in product(caracteres, repeat = i):
17                 prueba = "".join(comb)
18                 archivo.write(prueba+"\n")
19                 if prueba == con:
20                     print("La contraseña es {}".format(prueba))
21                     break
22             archivo.close()
23     else:
24         print("Ingresa una contraseña de longitud 3 o 4")
25
26 if __name__ == "__main__":
27     con = input("ingresa una contraseña\n")
28     t0 = time()
29     buscar(con)
30     print("Tiempo de ejecucion {}\n".format(round(time()-t0,6)))
31
```

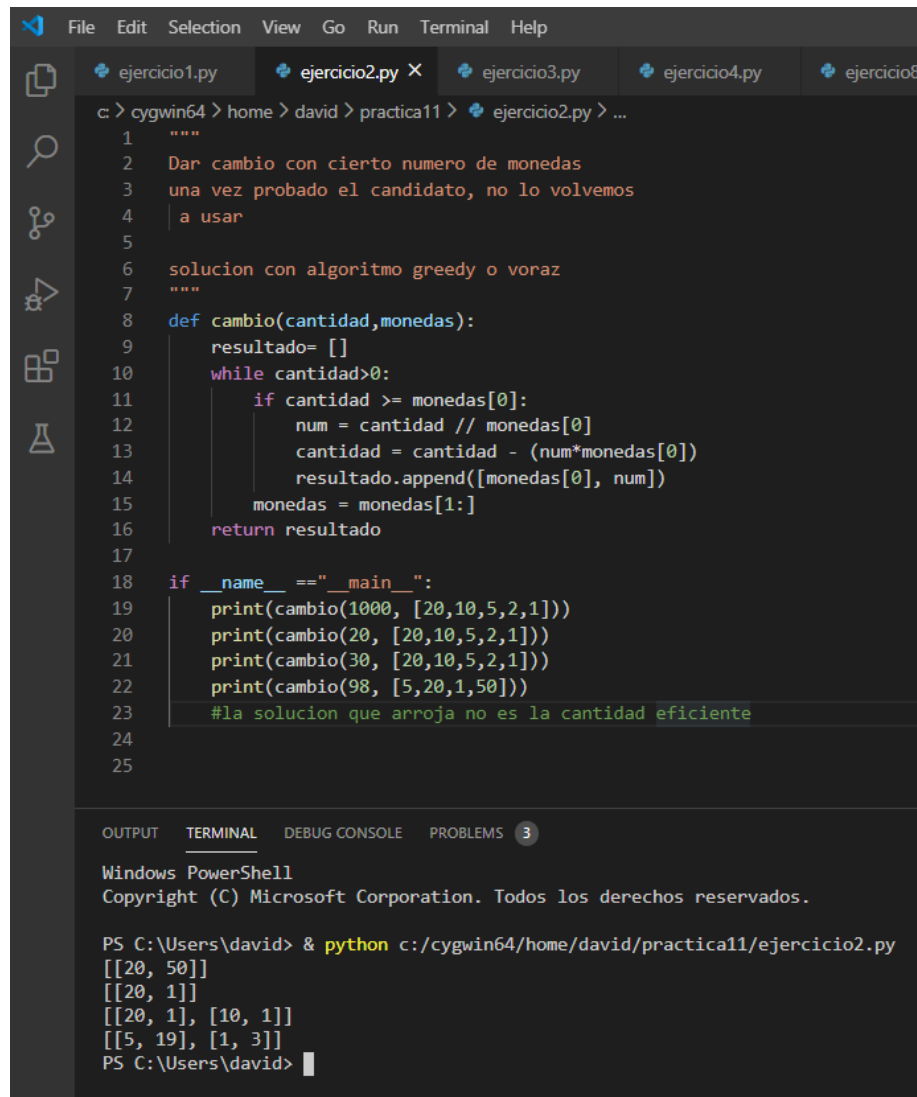
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 3

```
PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio1.py
ingresa una contraseña
ale
La contraseña es ale
Tiempo de ejecucion 19.77372

PS C:\Users\david>
PS C:\Users\david>
PS C:\Users\david>
```

■ PROGRAMA 2.

En el programa 2 se busca encontrar la manera más eficiente de dar cambio con ciertos tipos de valores en moneda a distintas cantidades, en este caso ocupamos un algoritmo greedy o también llamado voraz, el programa se encarga de dividir la cantidad que tenemos sin que, sobre dinero, así en cada moneda, todo esto por medio de la división entera

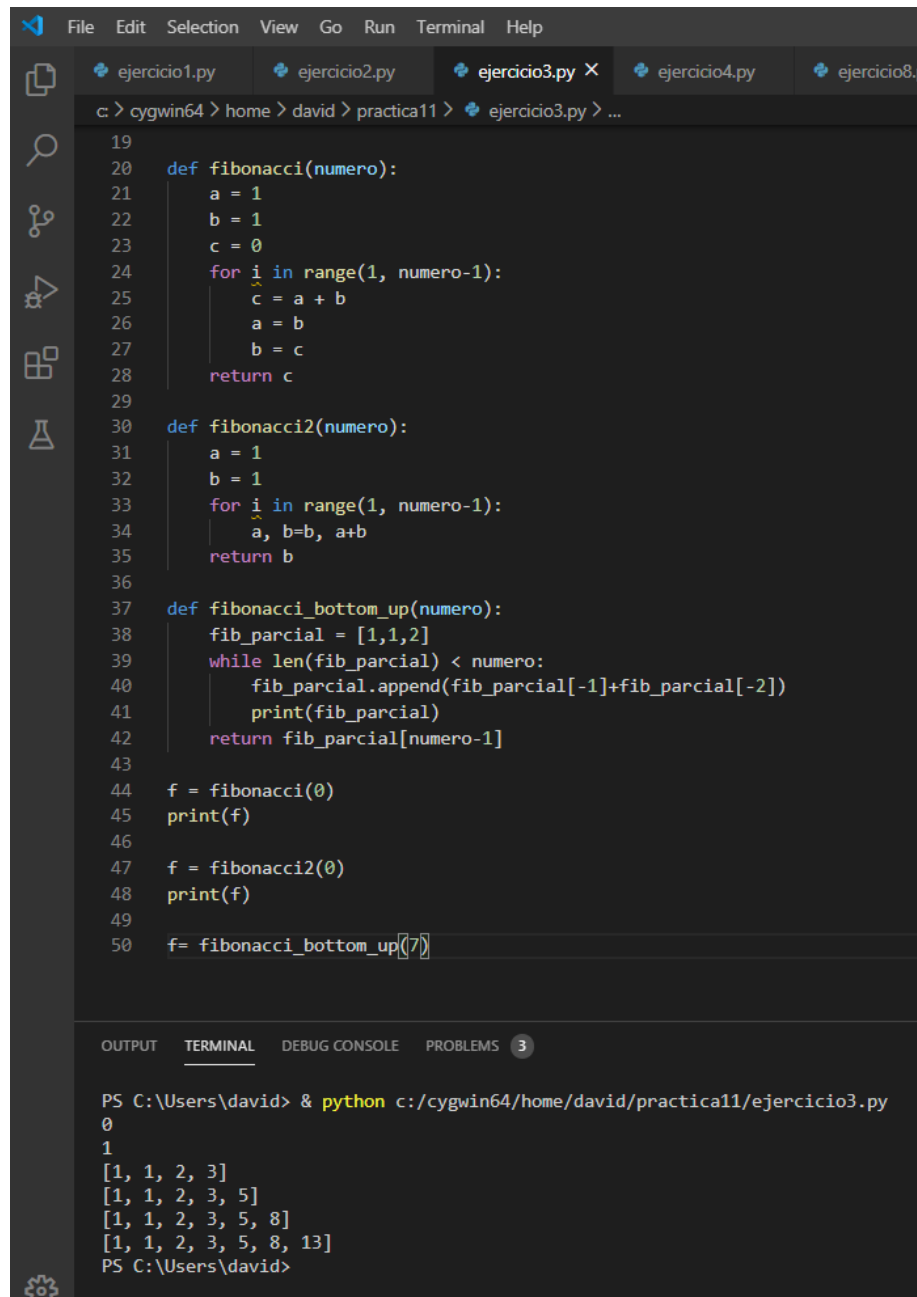


```
File Edit Selection View Go Run Terminal Help
ejercicio1.py ejercicio2.py X ejercicio3.py ejercicio4.py ejercicio8
c > cygwin64 > home > david > practica11 > ejercicio2.py > ...
1  """
2  Dar cambio con cierto numero de monedas
3  una vez probado el candidato, no lo volvemos
4  a usar
5
6  solucion con algoritmo greedy o voraz
7  """
8  def cambio(cantidad,monedas):
9      resultado= []
10     while cantidad>0:
11         if cantidad >= monedas[0]:
12             num = cantidad // monedas[0]
13             cantidad = cantidad - (num*monedas[0])
14             resultado.append([monedas[0], num])
15             monedas = monedas[1:]
16     return resultado
17
18 if __name__ == "__main__":
19     print(cambio(1000, [20,10,5,2,1]))
20     print(cambio(20, [20,10,5,2,1]))
21     print(cambio(30, [20,10,5,2,1]))
22     print(cambio(98, [5,20,1,50]))
23     #la solucion que arroja no es la cantidad eficiente
24
25
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 3
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio2.py
[[20, 50]]
[[20, 1]]
[[20, 1], [10, 1]]
[[5, 19], [1, 3]]
PS C:\Users\david>
```

PROGRAMA 3.

En este caso se nos pide encontrar la sucesión de Fibonacci, la cual como antecedente sabemos que se encuentra por la suma de los dos números anteriores, así en cada caso, esta vez lo hicimos por el algoritmo de forma ascendente, el cual busca la respuesta de abajo hacia arriba, obteniendo los primeros números y así ir obteniendo los siguientes hasta llegar a la respuesta que se necesita.



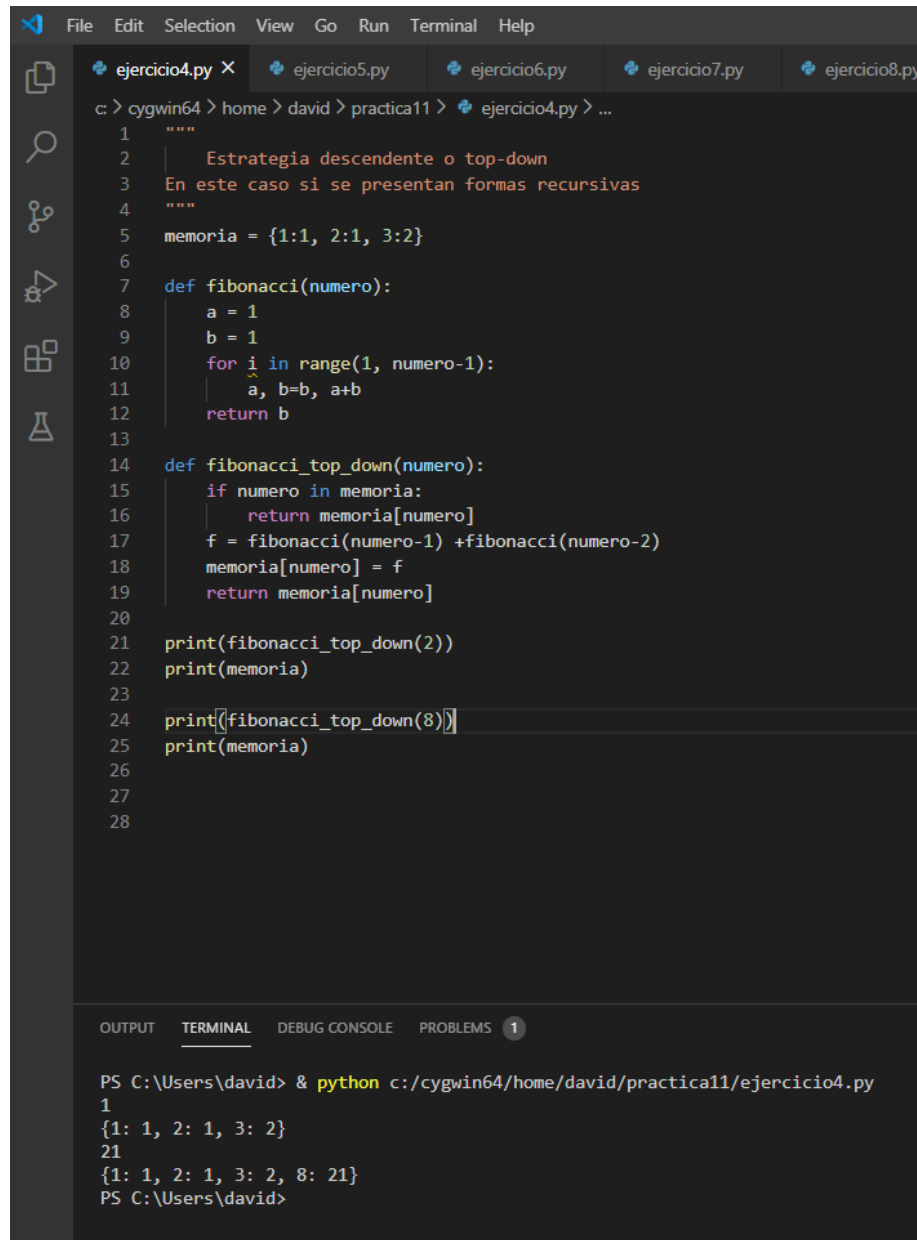
```
File Edit Selection View Go Run Terminal Help
ejercicio1.py ejercicio2.py ejercicio3.py X ejercicio4.py ejercicio8.py
c > cygwin64 > home > david > practica11 > ejercicio3.py > ...

19
20 def fibonacci(numero):
21     a = 1
22     b = 1
23     c = 0
24     for i in range(1, numero-1):
25         c = a + b
26         a = b
27         b = c
28     return c
29
30 def fibonacci2(numero):
31     a = 1
32     b = 1
33     for i in range(1, numero-1):
34         a, b = b, a+b
35     return b
36
37 def fibonacci_bottom_up(numero):
38     fib_parcial = [1,1,2]
39     while len(fib_parcial) < numero:
40         fib_parcial.append(fib_parcial[-1]+fib_parcial[-2])
41         print(fib_parcial)
42     return fib_parcial[numero-1]
43
44 f = fibonacci(0)
45 print(f)
46
47 f = fibonacci2(0)
48 print(f)
49
50 f= fibonacci_bottom_up(7)

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 3
PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio3.py
0
1
[1, 1, 2, 3]
[1, 1, 2, 3, 5]
[1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8, 13]
PS C:\Users\david>
```

■ PROGRAMA 4.

En este caso hicimos de nuevo la sucesión de Fibonacci, sólo que esta vez lo hicimos por medio de la estrategia top-down, la cual como indica el nombre es descendente, es decir que va buscando los casos más grandes y no se encarga prácticamente de los casos pequeños a menos de que se le indique, en este caso si se presentan formas recursivas. El programa es dentro de lo que cabe eficiente, ya que en realidad va directo a una respuesta, como se pude observar cuando lo programamos, pero no Podemos observar los casos anteriores al que pedimos.

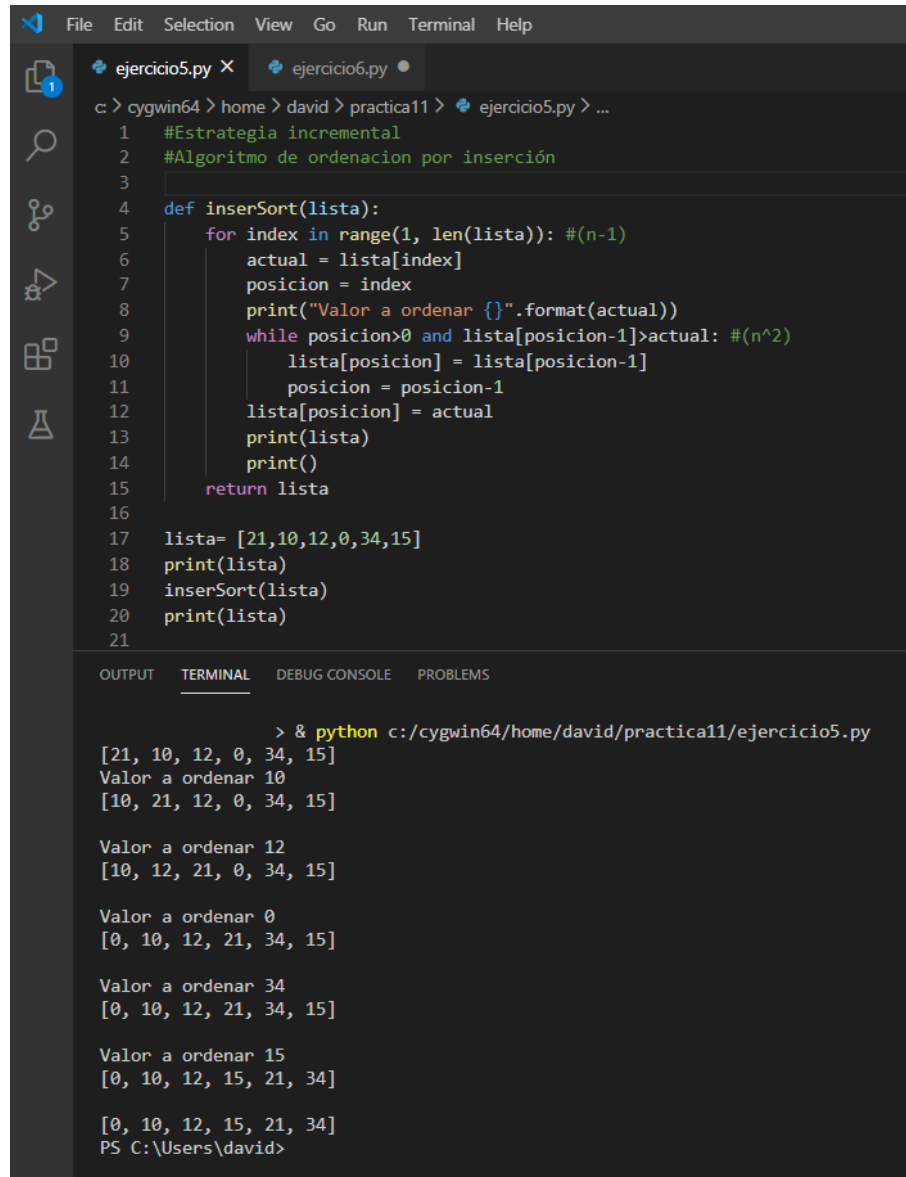


```
File Edit Selection View Go Run Terminal Help
ejercicio4.py X ejercicio5.py ejercicio6.py ejercicio7.py ejercicio8.py
c > cygwin64 > home > david > practica11 > ejercicio4.py > ...
1 """
2     Estrategia descendente o top-down
3     En este caso si se presentan formas recursivas
4 """
5 memoria = {1:1, 2:1, 3:2}
6
7 def fibonacci(numero):
8     a = 1
9     b = 1
10    for i in range(1, numero-1):
11        a, b=b, a+b
12    return b
13
14 def fibonacci_top_down(numero):
15     if numero in memoria:
16         return memoria[numero]
17     f = fibonacci(numero-1) + fibonacci(numero-2)
18     memoria[numero] = f
19     return memoria[numero]
20
21 print(fibonacci_top_down(2))
22 print(memoria)
23
24 print(fibonacci_top_down(8))
25 print(memoria)
26
27
28

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1
PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio4.py
1
{1: 1, 2: 1, 3: 2}
21
{1: 1, 2: 1, 3: 2, 8: 21}
PS C:\Users\david>
```

■ PROGRAMA 5.

En este programa se nos pide ordenar números en orden ascendente es decir del menor al mayor, todos están desordenados. En este caso se hace el acomodo por medio de la estrategia incremental o algoritmo de ordenación por inserción, lo que hace nuestro programa es tomar los números de nuestra lista e ir viendo si son o no menores o mayores a los que ya pasaron a tomar un lugar en la lista acomodada, si son menores que todos, pasara al inicio de la lista, si es mayor a todos pasará al final y en caso de que esté entre dos valores, tomará el lugar del mayor de esos números y aquel pasará al lugar consecutivo.



```
c:\cygwin64\home\david\practica11> ejercicio5.py ...
1  #Estrategia incremental
2  #Algoritmo de ordenacion por inserción
3
4  def insertSort(lista):
5      for index in range(1, len(lista)): #(n-1)
6          actual = lista[index]
7          posicion = index
8          print("Valor a ordenar {}".format(actual))
9          while posicion>0 and lista[posicion-1]>actual: #(n^2)
10             lista[posicion] = lista[posicion-1]
11             posicion = posicion-1
12             lista[posicion] = actual
13             print(lista)
14             print()
15         return lista
16
17 lista= [21,10,12,0,34,15]
18 print(lista)
19 insertSort(lista)
20 print(lista)
21
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
> & python c:/cygwin64/home/david/practica11/ejercicio5.py
[21, 10, 12, 0, 34, 15]
Valor a ordenar 10
[10, 21, 12, 0, 34, 15]

Valor a ordenar 12
[10, 12, 21, 0, 34, 15]

Valor a ordenar 0
[0, 10, 12, 21, 34, 15]

Valor a ordenar 34
[0, 10, 12, 21, 34, 15]

Valor a ordenar 15
[0, 10, 12, 15, 21, 34]

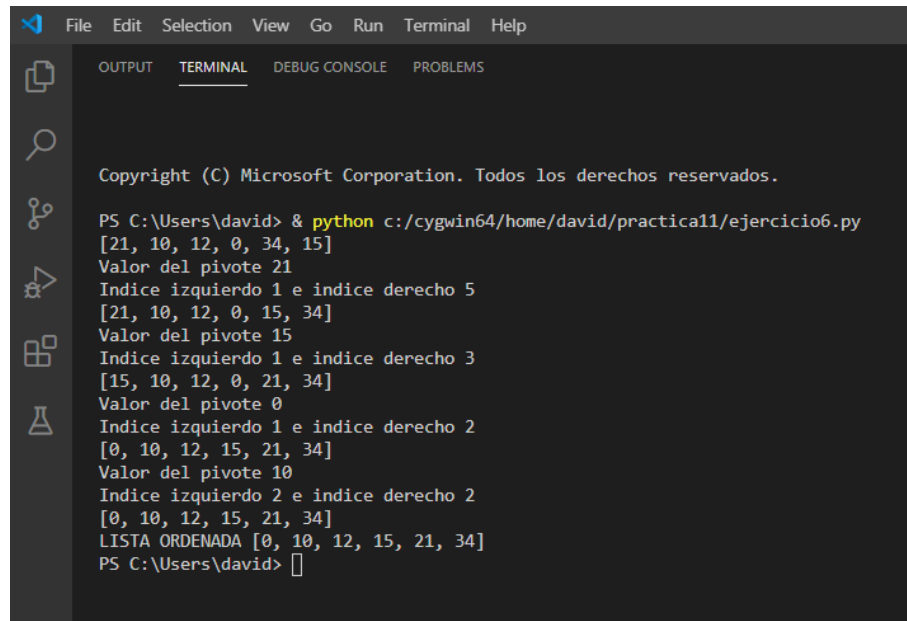
[0, 10, 12, 15, 21, 34]
PS C:\Users\david>
```

■ PROGRAMA 6.

En este programa se nos pide lo mismo que en el anterior, lo cual es ordenar una lista de números en orden ascendente, pero esta vez utilizaremos la estrategia divide y vencerás, lo que hace este programa es seleccionar un pivote, es importante seleccionar un buen pivote, ya que por medio de este se medirá la eficiencia de nuestro programa y realiza una división del programa

En este programa consideramos como pivote al inicio a un lado de el estará el “lado izquierdo” y al final de la lista tenemos al “lado derecho”, ahora si comenzamos a entrar en los ciclos del programa, ya que si el valor de la izquierda es menor al de la derecha y al mismo tiempo la parte izquierda es menor que el pivote incrementamos el índice de izquierda y mientras el valor de la derecha sea mayor al valor de la izquierda y la parte derecha sea mayor al pivote incrementamos el Contador del índice de derecha, ahora si también derecha es menor a izquierda, salimos del ciclo, ahora en caso de que no sea así tenemos una variable temporal que es la lista izquierda y si son iguales la parte izquierda con la derecha la variable temporal decimos que también la parte derecha es igual a la variable temporal y eso es igual a el inicio de nuestra lista, entonces regresamos la derecha.

```
File Edit Selection View Go Run Terminal Help
ejercicio6.py X
c > cygwin64 > home > david > practica11 > ejercicio6.py > particion
19 """
20 def quicksort(lista):
21     quicksort_2(lista,0,len(lista)-1)
22
23
24 def quicksort_2(lista, inicio, fin):
25     if inicio < fin:
26
27         pivote = particion(lista, inicio, fin)
28
29         quicksort_2(lista, inicio, pivote-1)
30         quicksort_2(lista, pivote+1, fin)
31
32 def particion(lista, inicio, fin):
33     pivote = lista[inicio]
34     print("Valor del pivote {}".format(pivote))
35     izquierda = inicio+1
36     derecha = fin
37     print("Indice izquierdo {} e indice derecho {}".format(izquierda, derecha))
38
39     bandera = False #las banderas nos indican si ya terminamos
40     while not bandera:
41         while izquierda <= derecha and lista[izquierda] <= pivote:
42             izquierda = izquierda + 1
43         while derecha >= izquierda and lista[derecha] >= pivote:
44             derecha = derecha -1
45         if derecha < izquierda:
46             bandera = True
47         else:
48             temp = lista[izquierda]
49             lista[izquierda] = lista[derecha]
50             lista[derecha] = temp
51
52     print(lista)
53
54     temp = lista[inicio]
55     lista[inicio] = lista[derecha]
56     lista[derecha] = temp
57     return derecha
58
59 lista=[21, 10, 12, 0, 11, 9, 24, 20, 14, 1]
60 print(lista)
61 quicksort(lista)
62 print("LISTA ORDENADA {}".format(lista))
63
```



The image shows a Visual Studio Code terminal window with a dark theme. The menu bar at the top includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The 'TERMINAL' tab is active, showing the output of a Python script. The script implements a quicksort algorithm, displaying the initial array, pivot selection, partitioning steps, and the final sorted array.

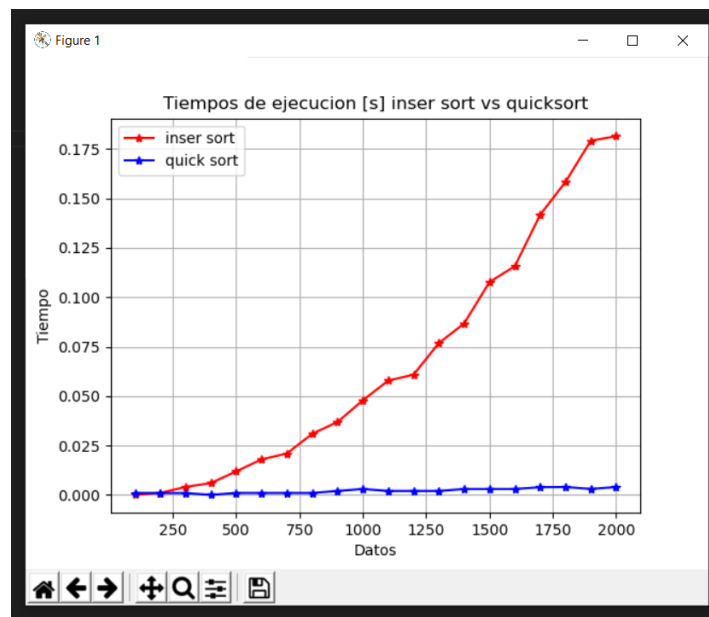
```
PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio6.py
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\david> & python c:/cygwin64/home/david/practica11/ejercicio6.py
[21, 10, 12, 0, 34, 15]
Valor del pivote 21
Indice izquierdo 1 e indice derecho 5
[21, 10, 12, 0, 15, 34]
Valor del pivote 15
Indice izquierdo 1 e indice derecho 3
[15, 10, 12, 0, 21, 34]
Valor del pivote 0
Indice izquierdo 1 e indice derecho 2
[0, 10, 12, 15, 21, 34]
Valor del pivote 10
Indice izquierdo 2 e indice derecho 2
[0, 10, 12, 15, 21, 34]
LISTA ORDENADA [0, 10, 12, 15, 21, 34]
PS C:\Users\david>
```


PROGRAMA 7

En este programa utilizamos bibliotecas para crear las gráficas de los ejercicios 5 y 6, por medio de ellas pudimos observar la eficiencia de cada uno de los algoritmos que creamos, ya que compara los tiempos de ejecución a cada dato que dan los programas.

```
File Edit Selection View Go Run Terminal Help
ejercicio5.py ejercicio6.py ejercicio7.py X ejercicio8.py
c:\cygwin64\home\david\practica11> ejercicio7.py > ...
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.mplot3d import Axes3D
3 import random
4 from time import time
5 from ejercicio5 import inserSort
6 from ejercicio6 import quicksort
7
8 datos = [ii*100 for ii in range(1,21)]
9
10 tiempo_is = []
11 tiempo_qs = []
12
13 for ii in datos:
14     lista_is = random.sample(range(0, 10000000), ii)
15     lista_qs = lista_is.copy()
16
17     t0 = time()
18     inserSort(lista_is)
19     tiempo_is.append(round(time()-t0,6)) #lista de tiempos que vamos a obtener
20
21     t0 = time()
22     quicksort(lista_qs)
23     tiempo_qs.append(round(time()-t0,6))
24
25 print("Tiempos parciales de ejecución en insertsort {} [s]".format(tiempo_is))
26 print("Tiempos parciales de ejecución en quicksort {} [s]".format(tiempo_qs))
27
28 fig, ax = plt.subplots()
29 ax.plot(datos, tiempo_is, label="inser sort", marker="*", color="r")
30 ax.plot(datos, tiempo_qs, label="quick sort", marker="*", color="b")
31
32 ax.set_xlabel("Datos")
33 ax.set_ylabel("Tiempo")
34 ax.grid(True)
35 ax.legend(loc=2)
36
37
38 plt.title("Tiempos de ejecucion [s] inser sort vs quicksort")
39 plt.show()
```



OUTPUTTERMINALDEBUG CONSOLEPROBLEMS

1: Python

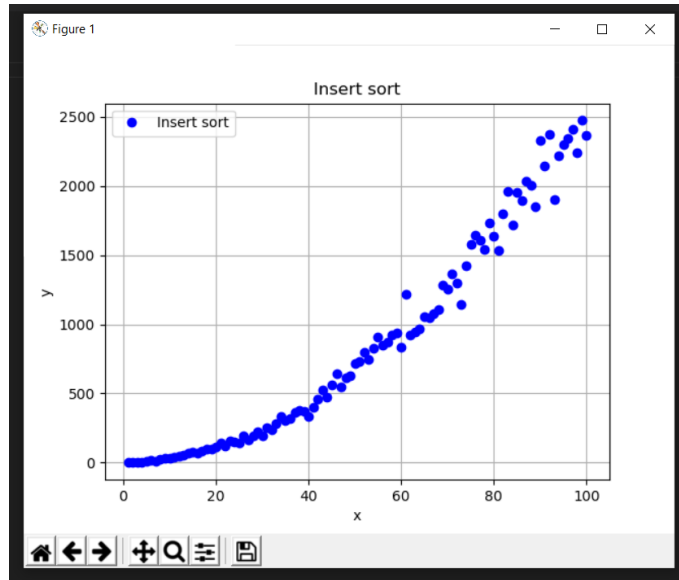
+[]⌵×

```
Tiempos parciales de ejecución en insertsort [0.0, 0.001013, 0.004044, 0.005994, 0.011919, 0.017951, 0.020944, 0.030916, 0.036901, 0.047872, 0.057844, 0.060837, 0.076795, 0.086768, 0.107715, 0.11569, 0.141739, 0.158342, 0.179201, 0.181487] [s]
Tiempos parciales de ejecución en quicksort [0.000998, 0.000984, 0.000944, 0.0, 0.000998, 0.000997, 0.000998, 0.000997, 0.001995, 0.002993, 0.001995, 0.001995, 0.001995, 0.001995, 0.002992, 0.002989, 0.002992, 0.003899, 0.004003, 0.003023, 0.003989] [s]
```

PROGRAMA 8

En este último programa creamos la gráfica de la función insertSort la cuál utilizamos en el ejercicio 5 para acomodar una lista, la gráfica es un análisis de complejidad utilizando el modelo RAM que contabiliza las veces que se realiza un ciclo en cierto tiempo.

```
File Edit Selection View Go Run Terminal Help
ejercicio5.py ejercicio6.py ejercicio7.py ejercicio8.py X
c > cygwin64 > home > david > practica11 > ejercicio8.py > ...
1  """
2  TAREA PRACTICA 10 GRAFICA
3  """
4  import matplotlib.pyplot as plt
5  from mpl_toolkits.mplot3d import Axes3D
6  import random
7
8  times = 0
9
10 def insertSort(lista):
11     global times
12     for i in range(1, len(lista)):
13         times +=1
14         actual = lista[i]
15         posicion = i
16         while posicion > 0 and lista[posicion-1] > actual:
17             times += 1
18             lista[posicion] = lista[posicion-1]
19             posicion = posicion-1
20         lista[posicion] = actual
21     return lista
22
23 TAM = 101
24 eje_x = list(range(1, TAM, 1)) #Va de uno en uno
25 eje_y = []
26
27 lista_variable = []
28
29 for num in eje_x:
30     lista_variable = random.sample(range(0, 1000), num)
31     times = 0
32     lista_variable = insertSort(lista_variable)
33     eje_y.append(times)
34
35 fig, ax = plt.subplots(facecolor='w', edgecolor='k')
36 ax.plot(eje_x, eje_y, marker="o", color="b", linestyle="None")
37
38 ax.set_xlabel('x')
39 ax.set_ylabel('y')
40 ax.grid(True)
41 ax.legend(["Insert sort"])
42
43 plt.title("Insert sort")
44 plt.show()
```



❖ CONCLUSIONES

- Esta práctica me parece importante pues es toda la otra parte del curso en el que se manejan los tipos de algoritmos que existen, además de que ver la eficiencia de cada una, no sólo calculándola por medio de estándares teóricos, sino que también me pareció importante hacer modelos con los cuales pudimos programar las gráficas que nos sirvieron más para englobar ese concepto de eficiencia en un programa.
- También me agrado ver en los programas como es que las propias ejecuciones de cada uno poco a poco pueden irte dando una mejor idea sobre el funcionamiento de cada algoritmo.