UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Introducción a Ciencia de la Computación

Ejercicios para Práctica Calificada 4

Las siguientes preguntas se recogen de prácticas pasadas y están acompañadas de un indicador de dificultad que va desde el 1 al 5.

Ordenamiento

1. (Nivel 2) Implemente una función para ordenar los datos que hay en una lista de listas, con la siguiente estructura: nombre, edad y altura. La función debe recibir dos parámetros: la lista de listas y la columna por la cual ordenar.

   Un ejemplo de los datos sería lo siguiente:

```
datos = [['Maria', 25, 1.60],
         ['Carlos', 31, 1.70],
         ['Jimena', 27, 1.65],
         ['Kike', 24, 1.69],
         ['Lucero', 23, 1.58],
         ['Roberto', 28, 1.64] ]
```

   Considere, que los datos ya están en el programa, no necesita ingresar la matriz de datos, y realize el siguiente proceso:

   - Solicite al usuario que ingrese por cual columna ordenar: Nombre, edad, altura
   - Ordenar los datos de forma ascendente, por la columna elegida
   - Imprimir los datos ordenadas en forma de tabla

   También calcule cuál es la complejidad del algoritmo implementado y grafique la curva de la complejidad.

   Algunos ejemplos de diálogo de este programa serían:

```
Por cual columna ordeno: nombre
Respuesta:
'Carlos', 31, 1.70
'Jimena', 27, 1.65
'Kike', 24, 1.69
'Lucero', 23, 1.58
'Maria', 25, 1.60
'Roberto', 28, 1.64
```

```
Por cual columna ordeno: altura
Respuesta:
'Lucero', 23, 1.58
'Maria', 25, 1.60
'Roberto', 28, 1.64
'Jimena', 27, 1.65
'Kike', 24, 1.69
'Carlos', 31, 1.70
```

2. (Nivel 2) Se tiene una lista de resultados de una entrevista a candidatos a profesores de una universidad. Cada entrevista ha considerado almacenarse en un diccionario que tiene la siguiente información: Nombre, edad, puntaje de clase modelo, grado obtenido y número de publicaciones indizadas.

   Implemente el algoritmo de ordenamiento de la lista de diccionarios de menor a mayor valor usando como criterio de ordenación el puntaje de la clase modelo. Complete la función ordenarlista e implemente su propio algoritmo de ordenamiento según los revisados en clase. Adicionalmente grafique la función de crecimiento $\mathcal{O}(n)$.

```
candidatos = [
{'nombre':'Juan Pablo', 'edad':25, 'puntaje':10, 'grado':'
   master', 'pub':1},
{'nombre':'Rocio Isabel', 'edad':32, 'puntaje':3, 'grado':'
   bachiller', 'pub':34},
{'nombre':'Maria Fernanda', 'edad':33, 'puntaje':14, 'grado
   ':'doctor', 'pub':1},
{'nombre':'Rodrigo', 'edad':53, 'puntaje':8, 'grado':'doctor
   ', 'pubs':9},
{'nombre':'Ernesto', 'edad':65, 'puntaje':12, 'grado':'
   master', 'pub':10}
]

def ordenarlista( candidatos ):
    # ToDO
    return asistentes
```

```
Output esperado:
[{'nombre':'Rocio Isabel', 'edad':32, 'puntaje':3, 'grado':'
   bachiller', 'pub':34},
{'nombre':'Rodrigo', 'edad':53, 'puntaje':8, 'grado':'doctor
   ', 'pub':9}
{'nombre':'Juan Pablo', 'edad':25, 'puntaje':10, 'grado':'
   master', 'pub':1},
,{'nombre':'Ernesto', 'edad':65, 'puntaje':12, 'grado':'
   master', 'pub':10},
```

```
{'nombre':'Maria Fernanda', 'edad':33, 'puntaje':14, 'grado
   ':'doctor', 'pubs':1}]
```

3. (Nivel 2) Dado un archivo de texto `turitin.txt`, donde cada fila representa la información de un usuario con los siguientes atributos:

   - nombre,apellidopaterno,apellidomaterno,sueldo

   Se pide lo siguiente:

   - Leer el archivo de texto y crear una lista, donde cada elemento de la lista sea a su vez una lista, cuyos datos sean los datos de cada usuario.
   - Ordenar la lista por sueldo de manera descendente utilizando un algoritmo de ordenamiento y luego imprimirla.
   - Determinar la eficiencia del algoritmo y diagramarla en un plano cartesiano.

   Algunos ejemplos de diálogo de este programa serían:

```
Input: turitin.txt
mariano,melgar,zavala,5000
pablo,macorito,romero,10000
felipa,gertron,hurtado,1230

Output: lista con sublistas ordenadas por sueldo de forma
   descendente:
[[pablo,macorito,romero,10000],[mariano,melgar,zavala
   ,5000],[felipa,gertron,hurtado,1230]]
```

4. (Nivel 2) Dada la siguiente tabla de productos lanzados para la Navidad de 2019:

nombre	contenido	precio
Panettone	frutas	15
Chocoton	chocolate	13
Camottone	camote	18
Oreotone	oreo	40
Emeone	mandm	30
Stollen	frutas	20
Alfajorone	manjarblanco	11

   - Crear una lista de diccionarios con los datos de la tabla, donde cada fila de la tabla es un diccionario.
   - Ordenar la lista por precio de forma ascendente, usando un algoritmo estudiado en clase.
   - Imprimir la lista ordenada.

- Indicar la función de complejidad del algoritmo de ordenamiento implementado y graficarla.

Algunos ejemplos de diálogo de este programa serían:

```
Output:
[{'nombre': 'Alfajorone', 'contenido': 'manjarblanco', '
   precio': 11},
{'nombre': 'Chocoton', 'contenido': 'chocolate', 'precio':
   13},
{'nombre': 'Panettone', 'contenido': 'frutas', 'precio':
   15},
{'nombre': 'Camottone', 'contenido': 'camote', 'precio':
   18},
{'nombre': 'Stollen', 'contenido': 'frutas', 'precio': 20},
{'nombre': 'Emeone', 'contenido': 'mandm', 'precio': 30},
{'nombre': 'Oretone', 'contenido': 'oreo', 'precio': 40}]
```

5. (Nivel 3) Implemente una función para ordenar los datos que hay en una lista de listas de estrellas, con la siguiente estructura: Nombre, distancia (años luz), Magnitud V; la función debe tomar dos parámetros: la lista de listas y la columna por cual ordenar.

   Un ejemplo sería, la lista de estrellas más brillantes:

```
datos = [['Sol', 0.000016, -26.73],
        ['Sirio', 8.6, -1.47],
        ['Canopus',310, -0.72],
        ['Rigil␣Kentaurus␣A', 4.4, -0.27],
        ['Arturo', 37, -0.04],
        ['Vega', 25, 0.03],
        ['Rigel', 770, 0.12],
        ['Procyon', 11, 0.34],
        ['Achernar', 140, 0.50]]
```

   Considere, que los datos ya están en el programa, no necesita ingresar la matriz de datos, y realize el siguiente proceso:

   - Solicite al usuario que ingrese la columna a ordenar: Nombre, distancia, magnitud
   - Ordenar los datos de forma ascendente, por la columna elegida
   - Imprimir los datos ordenados en forma de tabla

   También calcule cuál es la complejidad del algoritmo implementado y grafique la curva de la complejidad. Algunos ejemplos de diálogo de este programa serían:

```
Ordenar por la columna: nombre
Nombre, distancia, magnitud
'Achernar', 140, 0.50
```

```
'Arturo', 37, -0.04,
'Canopus',310, -0.72,
'Procyon', 11, 0.34,
'Rigel', 770, 0.12,
'Rigil␣Kentaurus␣A', 4.4, -0.27,
'Sirio', 8.6, -1.47,
'Sol', 0.000016, -26.73,
'Vega', 25, 0.03,
```

```
Ordenar por la columna: magnitud
Nombre, distancia, magnitud
'Sol', 0.000016, -26.73,
'Sirio', 8.6, -1.47,
'Canopus',310, -0.72,
'Rigil␣Kentaurus␣A', 4.4, -0.27,
'Arturo', 37, -0.04,
'Vega', 25, 0.03,
'Rigel', 770, 0.12,
'Procyon', 11, 0.34,
'Achernar', 140, 0.50
```

6. (Nivel 3) Dado un archivo de texto `file.txt`, donde cada fila representa la información de un usuario con las siguientes atributos:

   - nombre
   - edad
   - sexo
   - hobby

   Se pide lo siguiente:

   - Leer el archivo de texto y crear un diccionario para cada información del usuario y posteriormente agregarlos a una lista.
   - Ordenar la lista por edad utilizando un algoritmo de ordenamiento.
   - Determinar la eficiencia del algoritmo y diagramarlo en un plano cartesiano donde la abscisa (eje x) es el número de entrada y la ordenada (eje y) es el tiempo de ejecución del algoritmo.

   Algunos ejemplos de diálogo de este programa serían:

```
Input: file.txt
marco 21 M futbol
jorge 26 M baskquet
teresa 18 F volley
sergio 48 M tenis
```

```
mario 41 M atletismo
enrique 18 M natacion

Output: Diccionarios ordenados por edad en la Lista:
[{'edad': 18, 'hobby': 'volley', 'nombre': 'teresa', 'sexo':
    'F'},
 {'edad': 18, 'hobby': 'natacion', 'nombre': 'enrique', '
    sexo': 'M'},
 {'edad': 21, 'hobby': 'futbol', 'nombre': 'marco', 'sexo':
    'M'},
 {'edad': 26, 'hobby': 'baskquet', 'nombre': 'jorge', 'sexo'
    : 'M'},
 {'edad': 41, 'hobby': 'atletismo', 'nombre': 'mario', 'sexo
    ': 'M'},
 {'edad': 48, 'hobby': 'tenis', 'nombre': 'sergio', 'sexo':
    'M'}]
```

7. (Nivel 3) En UTEC se ha definido un criterio para comparar matrices: una matriz será mayor, menor o igual si la suma de sus elementos es mayor, menor o igual que la suma de los elementos de la segunda matriz. Implemente las funciones necesarias que permitan ordenar una lista de matrices bajo el criterio UTEC. Adicionalmente, indique cuál es la complejidad algorítmica del método implementado, grafique la curva de la misma e indique qué factores pueden afectar el tiempo del algoritmo. Considere que su algoritmo debe funcionar para una lista de cualquier tamaño que contenga matrices de cualquier tamaño. No es necesario implementar el ingreso de la información hacia la lista o matrices.

   Algunos ejemplos de diálogo de este programa serían:

```
Input: [[[17, 15], [2, 2]], [[0, 2, 3], [4, 4, 4]], [[4, 0],
    [5, 0]]]
Output: [[[4, 0], [5, 0]], [[0, 2, 3], [4, 4, 4]], [[17,
   15], [2, 2]]]
```

```
Input: [[[1, 1], [2, 25]], [[8, 2, 3], [4, 4, 4]], [[4, 4],
   [5, 5]]]
Output: [[[4, 4], [5, 5]], [[8, 2, 3], [4, 4, 4]], [[1, 1],
   [2, 25]]]
```

8. (Nivel 3) En UTEC se ha definido un criterio para comparar listas: una lista será mayor, menor o igual que otra dependiendo si el promedio de sus elementos es mayor, menor o igual que el promedio de la segunda lista. Bajo este criterio, implemente las funciones necesarias que permitan ordenar una lista de listas de menor a mayor. Adicionalmente, indique cuál es la complejidad algorítmica del método implementado, grafique la curva de la misma e indique qué factores pueden afectar el desempeño del algoritmo. Considere

que su algoritmo debe funcionar para una lista de cualquier tamaño que contenga listas de cualquier tamaño. No es necesario implementar el ingreso de la información hacia las listas.

Algunos ejemplos de diálogo de este programa serían:

```
Input: [[5, 5], [1, 2, 3], [2, 4, 0, 0]]
Output: [[2, 4, 0, 0], [1, 2, 3], [5, 5]]
```

```
Input: [[2, 4], [3, 3], [1, 1, 0, 0], [4, 0]]
Output: [[1, 1, 0, 0], [4, 0], [2, 4], [3, 3]]
```

9. (Nivel 4) El método de los K vecinos más cercanos es usado en muchas aplicaciones de inteligencia artificial para tareas de clasificación y/o predicción. El algoritmo consiste en dado un conjunto de n elementos $(P1, P2, P3, ..., Pn)$ y un elemento de consulta Q, buscar los k elementos Pi que sean más cercanos a Q en función a una medida de distancia. En este apartado, nosotros vamos aplicar dicho algoritmo sobre un grupo grande de personas para hallar los k individuos cuyas edades son las más cercanas a una edad de consulta. Considerar como medida de distancia entre dos edades, el valor absoluto de la diferencia de ambas: $distancia(Pi, Q) = |Pi - Q|$.

   Un ejemplo de los datos de entrada y salida del programa son:

```
Lista de edades: 16,17,21,18,25,15
Edad de consulta: 19
K vecinos: 3
Resultado:
(18, 1)
(17, 2)
(21, 2)
```

   Como resultado se imprime línea por línea las k edades más cercanas a la edad de consulta con su respectiva distancia en formato de tupla $(edad, distancia)$.

   El prototipo del programa lleva la siguiente estructura:

```
def algoritmoOrdenamiento(listaTuplas):
  # Escriba aqui cualquier algoritmo de ordenamiento
  # Ejemplo: listaTuplas = [(16, 3), (17, 2), (21, 2), ...]

def juntarEdadDistancia(lista, Q):
  # Escriba aquí el algoritmo para crear una nueva lista
  # asociando cada edad con su distancia a la edad Q

def K_VecinosCercanos(lista, Q, k):
  # Realiza la búsqueda de los k más cercanos
  # Ejemplo: lista = [16,17,21,18,25,15], Q = 19, k = 3
  listaTuplas = juntarEdadDistancia(lista, Q)
```

```
  algoritmoOrdenacion(listaTuplas)
  return listaTuplas[:k]

#-----<lectura de datos y llamada a funciones>----
edadesStr = input("").split(",")
edadesInt = [int(i) for i in edadesStr]
Q = int(input(""))
k = int(input(""))
vecinos = K_VecinosCercanos(edadesInt, Q, k)
for e in vecinos:
  print(e)
```

Búsqueda Binaria

1. (Nivel 1) Tenemos un programa que contiene una lista cuyos elementos son dicccionarios de la siguiente forma:

   - actor, nacionalidad, película

   Ejemplo de una posible lista

```
actores=[ {"actor": "Felipe","nacionalidad":"peruana","
   pelicula":"loco␣por␣mary"},
 {"actor": "Gorilon","nacionalidad":"brasilera","pelicula":"
    asesinos␣locos"},
 {"actor": "Grecia","nacionalidad":"cubana","pelicula":"
    salvando␣al␣soldado␣Jacinto"},
 {"actor": "Teresita","nacionalidad":"venezolana","pelicula"
    :"tres␣y␣dos␣son␣ocho"}]
```

   Se pide lo siguiente:

   - Pedir el usuario que nos diga el actor a buscar.
   - Encontrar el actor usando un algoritmo de búsqueda binaria y mostrar todos sus datos.
   - Determinar la eficiencia del algoritmo y diagramarlo en un plano cartesiano.

   Algunos ejemplos de diálogo de este programa serían:

```
Ingrese nombre de actor a buscar: Felipe

Resultado:
nombre: Felipe
nacionalidad: peruana
película: loco por mary
```

```
Ingrese nombre de actor a buscar: Teresita

Resultado:
nombre: Teresita
nacionalidad: venezolana
película: tres y dos son ocho
```

2. (Nivel 2) Implemente una función, utlizando el algoritmo de busqueda binaria, para buscar datos que están en una lista de diccionarios sobre pel'iculas, con la siguiente estructura: titulo y año. La función debe recibir un parámetro (el año a buscar) y debe devolver el título de la película.

   Un ejemplo de una lista sería:

```
peliculas = [
        {'titulo': 'Shrek', 'año': 2001},
        {'titulo': 'El␣viaje␣de␣Chihiro', 'año': 2002},
        {'titulo': 'Buscando␣a␣Nemo', 'año': 2003},
        {'titulo': 'Los␣Increíbles', 'año': 2004},
        {'titulo': 'Wallace␣y␣Gromit', 'año': 2005},
        {'titulo': 'Happy␣Feet', 'año': 2006},
        {'titulo': 'Ratatouille', 'año': 2007},
        {'titulo': 'WALL-E', 'año': 2008},
        {'titulo': 'Up', 'año': 2009},
        {'titulo': 'Toy␣Story␣3', 'año': 2010},
        {'titulo': 'Rango', 'año': 2011},
        {'titulo': 'Valiente␣', 'año': 2012},
        {'titulo': 'Frozen', 'año': 2013},
        {'titulo': 'Grandes␣Héroes', 'año': 2014} ]
```

   Considere, que los datos ya están en el programa, no necesita ingresar la lista de películas, y realize el siguiente proceso:

   - Solicite al usuario que ingrese un año,
   - Buscar la película por el año, utilizando la función implementada,
   - Imprimir el título de la película.

   También calcule cuál es la complejidad del algoritmo implementado y grafique la curva de la complejidad. Algunos ejemplos de diálogo de este programa serían:

```
Ingrese año: 2002
Pelicula: El viaje de Chihiro
```

```
Ingrese año: 2013
Pelicula: Frozen
```

3. (Nivel 2) Implemente una función, utlizando el algoritmo de búsqueda binaria, para buscar datos que están en una lista de diccionarios sobre ganadores de premios nobel, con la siguiente estructura: año y ganadores. La función debe tomar un parámetro: el año a buscar y debe devolver los ganadores.

   Un ejemplo sería, la lista de ganadores de premios noble de Física:

```
peliculas = [
{'año':2010, 'nombre':'Andre␣Geim␣y␣Konstantín␣Novosiólov'},
{'año':2011, 'nombre':'Perlmutter,␣Schmidt␣y␣Adam␣Riess'},
{'año':2012, 'nombre':'Serge␣Haroche␣y␣David␣Wineland'},
{'año':2013, 'nombre':'Peter␣Higgs␣y␣François␣Englert'},
{'año':2014, 'nombre':'Akasaki,␣Hiroshi␣Amano␣y␣Nakamura'},
{'año':2015, 'nombre':'Takaaki␣Kajita␣y␣Arthur␣B.␣McDonald'
   },
{'año':2016, 'nombre':'Thouless,␣Haldane␣y␣Kosterlitz'},
{'año':2017, 'nombre':'Rainer␣Weiss,␣Barry␣Barish␣y␣Thorne'
   },
{'año':2018, 'nombre':'Donna␣Strickland,␣Mourou␣y␣Ashkin'},
{'año':2019, 'nombre':'James␣Peebles,␣Mayor␣y␣Queloz'} ]
```

   Considere, que los datos ya están en el programa, no necesita ingresar la lista, y realize el siguiente proceso:

   - Solicite al usuario que ingrese un año,
   - Buscar en la lista de diccionarios por el año, utilizando la función implementada,
   - Imprimir los nombres de los ganadores.

   También calcule cuál es la complejidad del algoritmo implementado y grafique la curva de la complejidad. Algunos ejemplos de diálogo de este programa serían:

```
Ingrese año: 2011
Ganadores: Perlmutter, Schmidt y Adam Riess
```

```
Ingrese año: 2018
Ganadores: Donna Strickland, Mourou y Ashkin
```

4. (Nivel 2) Se está implementando un programa para ESSALUD. Usted debe implementar el algoritmo de búsqueda binaria que devuelve los datos de un paciente que coincida con el DNI ingresado. Complete la función busbin. Calcule la complejidad algorítmica $\mathcal{O}(n)$ y grafique su función.

```
expedientes = [
{'nombre':'juan', 'tipo':'rh-', 'dni':11489834},
{'nombre':'maria', 'tipo':'a+', 'dni':12832055},
{'nombre':'jose', 'tipo':'a-', 'dni':12832055},
{'nombre':'paola', 'tipo':'o+','dni':12869666},
```

```
{'nombre':'gabriel', 'tipo':'o+', 'dni':13869179},
{'nombre':'maria', 'tipo':'o+','dni':16019959},
{'nombre':'susana', 'tipo':'rh-', 'dni':16426202},
{'nombre':'sofia', 'tipo':'rh-', 'dni':18947201},
{'nombre':'manuel', 'tipo':'b-', 'dni':18947201'},
{'nombre':'elene', 'tipo':'b+', 'dni':25480016},
{'nombre':'javier', 'tipo':'rh-', 'dni':28678360},
{'nombre':'rosario', 'tipo':'rh-', 'dni':28717605},
{'nombre':'jhuly', 'tipo':'rh-', 'dni':29600532},
{'nombre':'samuel', 'tipo':'rh-', 'dni':29718806}
]

dni = int(input("ingrese DNI a buscar: "))

def busbin(dni):
    #ToDO
    return k

dic_paciente = busbin(dni)

print("nombre:", dic_paciente['nombre'], ", tipo de sangre:
   ", dic_paciente['tipo'])
```

```
input:
29718806

output:
nombre: rosario, tipo de sangre: rh-
```

5. (Nivel 3) En UTEC se ha definido un criterio para comparar listas: una lista será mayor, menor o igual que otra dependiendo si el promedio de sus elementos es mayor, menor o igual que el promedio de la segunda lista. Implemente las funciones necesarias de tal forma que sea posible realizar búsquedas binarias iterativamente y así determinar si una lista determinada se encuentra o no en una lista de listas bajo el criterio definido por UTEC. Adicionalmente, indique cuál es la complejidad algorítmica del método implementado, grafique la curva de la misma e indique qué factores pueden afectar el desempeño del algoritmo. Considere que su algoritmo debe funcionar para una lista de cualquier tamaño que contenga listas de cualquier tamaño. No es necesario implementar el ingreso de la información hacia la lista o matrices. Asuma que la lista ya está ordenada.

   Algunos ejemplos de diálogo de este programa serían:

```
Input:[ [ [1, 1, 2, 2] ], [ [1, 2, 3, 4, 4, 4] ], [ [4, 4,
   5, 5] ] ]
Buscar: [ [3, 3] ]
```

```
Output: True
```

```
Input:[ [ [1, 1, 2, 2] ], [ [1, 2, 3, 4, 4, 4] ], [ [4, 4,
   5, 5] ] ]
Buscar: [ [3, 1] ]
Output: False
```

6. (Nivel 3) En UTEC se ha definido un criterio para comparar matrices: una matriz será mayor, menor o igual si la suma de sus elementos es mayor, menor o igual que la suma de los elementos de la segunda matriz. Implemente las funciones necesarias de tal forma que sea posible realizar búsqueda binaria recursivamente y así determinar si una matriz determinada se encuentra o no en una lista de matrices bajo el criterio definido por UTEC. Adicionalmente, indique cuál es la complejidad algorítmica del método implementado, grafique la curva de la misma e indique qué factores pueden afectar el desempeño del algoritmo. Considere que su algoritmo debe funcionar para una lista de cualquier tamaño que contenga matrices de cualquier tamaño. No es necesario implementar el ingreso de la información hacia la lista o matrices. Asuma que la lista ya está ordenada.

   Algunos ejemplos de diálogo de este programa serían:

```
Input:[ [ [1, 1], [2, 2] ], [ [0, 2, 3], [4, 4, 4] ], [ [4,
   4], [5, 5] ] ]
Buscar: [ [0, 0], [3, 3] ]
Output: True
```

```
Input:[ [ [1, 1], [2, 2] ], [ [0, 2, 3], [4, 4, 4] ], [ [4,
   4], [5, 5] ] ]
Buscar: [ [1, 4], [3, 3] ]
Output: False
```

7. (Nivel 4) Dado los siguientes diccionarios relacionados a los planetas del sistema planetario solar, el cual contiene los siguientes atributos:

   - planeta
   - diámetro

```
mercurio = {
    "planeta": "mercurio",
    "diametro": 4.878
}

venus = {
    "planeta": "venus",
    "diametro": 12.104
}
```

```
tierra = {
    "planeta": "tierra",
    "diametro": 12.756
}

marte = {
    "planeta": "marte",
    "diametro": 6.794
}

jupiter = {
    "planeta": "jupiter",
    "diametro": 142.800
}

saturno = {
    "planeta": "saturno",
    "diametro": 120.660
}

urano = {
    "planeta": "urano",
    "diametro": 51.800
}

neptuno = {
    "planeta": "neptuno",
    "diametro": 49.500
}
```

Se pide lo siguiente:

- Agregar los diccionarios a una lista
- Ordenar la lista por diámetro usando un algoritmo de ordenamiento.
- Encontrar el planeta con diámetro igual a `12.104` usando el algoritmo de búsqueda binaria.
- Determinar la eficiencia del algoritmo y diagramarlo en un plano cartesiano donde la abscisa (eje x) es el número de entrada y la ordenada (eje y) es el tiempo de ejecución del algoritmo.

Algunos ejemplos de diálogo de este programa serían:

```
Lista:
[{'diametro': 4.878, 'planeta': 'mercurio'},
```

```
 {'diametro': 12.104, 'planeta': 'venus'},
 {'diametro': 12.756, 'planeta': 'tierra'},
 {'diametro': 6.794, 'planeta': 'marte'},
 {'diametro': 142.8, 'planeta': 'jupiter'},
 {'diametro': 120.66, 'planeta': 'saturno'},
 {'diametro': 51.8, 'planeta': 'urano'},
 {'diametro': 49.5, 'planeta': 'neptuno'}]

Output:
Lista ordenada:
[{'diametro': 4.878, 'planeta': 'mercurio'},
 {'diametro': 6.794, 'planeta': 'marte'},
 {'diametro': 12.104, 'planeta': 'venus'},
 {'diametro': 12.756, 'planeta': 'tierra'},
 {'diametro': 49.5, 'planeta': 'neptuno'},
 {'diametro': 51.8, 'planeta': 'urano'},
 {'diametro': 120.66, 'planeta': 'saturno'},
 {'diametro': 142.8, 'planeta': 'jupiter'}]

El planeta con diámetro igual a 12.104 es venus
```

8. (Nivel 4) Escribir un programa que realize la búsqueda de un número entero dentro de los datos contenidos en un archivo matriz.txt:

```
1 2 3 4
5 6 7 8
9 10 11 12
```

Indicaciones:

- El usuario ingresa el número entero que desea buscar en la matriz.
- Abrir el archivo matriz.txt, y crear **una matriz** con los valores leídos.
- Usar **búsqueda binaria**.
- Imprimir la ubicación (fila, columna) en caso el elemento fuera encontrado.
- Caso contrario, mostrar el mensaje "El número no existe en la matriz".
- Indicar la función de complejidad del algoritmo de búsqueda implementado y graficarla.

Algunos ejemplos de diálogo de este programa serían:

```
Ingrese el número a buscar en la matriz: 4
El número se encuentra en (fila, columna): 1, 4
```

```
Ingrese el número a buscar en la matriz: 13
El número no existe en la matriz.
```

9. (Nivel 5) Se plantea el siguiente juego: en una mesa se tiene una fila de n vasos y dentro de cada vaso hay una bola enumerada. Se pide hallar la mínima cantidad de vasos que se necesita levantar para verificar la existencia de un número dado. Asumir que las bolas están ordenadas de forma descendente por lo tanto, lo mejor es aplicar la estrategia de búsqueda binaria que minimice el número de vasos a levantar.

Un ejemplo de los datos de entrada y salida del programa son:

```
Bolitas: 21,18,15,11,9,8,5,2
Buscar: 5
Vasitos levantados: 3
```

El prototipo del programa lleva la siguiente estructura:

```
def contarVasosLevantados(lista, search):
  cont = 0

  # Use el algoritmo de búsqueda binaria para
  # buscar el numero "search" y contabilice
  # la cantidad de vasitos a levantar en la variable "cont"

  return cont

bolitasStr=input("").split(",")
search=int(input(""))
bolitasInt = [int(i) for i in bolitasStr]
cont = contarVasosLevantados(bolitasInt, search)
print(cont)
```

Se le pide lo siguiente:

- Implementar la función para contar vasitos a levantar.
- Estime la cantidad de vasitos a levantar si se tiene las siguientes cantidades de elementos: 10, 100, 1000, 10000. Grafique la curva de la complejidad.