**Resumen**

**Integrantes: Luis Felipe Urquijo Vargas y Andres Prada**

Una de las aplicaciones más frecuentes, en programación es la ordenación.

• Existen dos técnicas de ordenación fundamentales en gestión de datos: ordenación de listas y ordenación de archivos.

• Los datos se pueden ordenar en orden ascendente o en orden descendente.

• Cada recorrido de los datos durante el proceso de ordenación se conoce como pasada o iteración.

• Los algoritmos de ordenación básicos son:

• Selección.

• Inserción.

• Burbuja.

• Los algoritmos de ordenación más avanzados son:

• Shell.

• Mergesort.

• Quicksort.

• La eficiencia de los algoritmos de burbuja, inserción y selección es 0(n2).

• La eficiencia de los algoritmos heapsort, radixsort, mergesort y quicksort es 0(n Log n).

• La búsqueda es el proceso de encontrar la posición de un elemento destino dentro de una lista. • Existen dos métodos básicos de búsqueda en arrays: búsqueda secuencial y binaria.

• La búsqueda secuencial se utiliza normalmente cuando el array no está ordenado. Comienza en el principio del array y busca hasta que se encuentra el dato buscado y se llega al final de la lista.

• Si un array está ordenado, se puede utilizar un algoritmo más eficiente denominado búsqueda binaria.

• La eficiencia de una búsqueda secuencial es 0(n).

• La eficiencia de una búsqueda binaria es 0(log n).

**Taller**

1. (10)Se desea eliminar todos los números duplicados de una lista

Por ejemplo

Si se le ingresan los valores [4,7,11,4,9,5,11,7,3,5]

Se debe cambia a [4,7,11,9,5,3]

import time

start = time.time()

lista = [2,3,4,5,6,4,5,6,2,3,3,5]

sin\_duplicados = []

miLista = list(lista)

for n in miLista:

    if n not in sin\_duplicados:

        sin\_duplicados.append(n)

    else:

        lista.remove(n)

print(lista)

print(sin\_duplicados)

end = time.time()

print(end-start)

1. (10)Elimine los elementos duplicados de un vector ordenado.

¿Cuál es la eficiencia del método?

Compárela con la eficiencia del punto 1

import time

start = time.time()

vector = [2,2,3,3,3,4,4,5,5,5,6,6]

sin\_duplicados = []

miVector = list(vector)

for n in miVector:

    if n not in sin\_duplicados:

        sin\_duplicados.append(n)

    else:

        vector.remove(n)

print(vector)

print(sin\_duplicados)

end = time.time()

print(end-start)

Algoritmo punto 1

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

Algoritmo punto 2

Texto

Descripción generada automáticamente

El tiempo de eficiencia del algoritmo 2 es mejor con respecto al primero.

1. (5)Dada la siguiente lista

[47,3,21,32.56,92]

Después de 2 “pasadas” de un algoritmo de ordenación, la lista ha quedado dispuesto asi

[3,21,47,32,56,92]

¿Qué algoritmo de ordenación se esta utilizando (selección, burbuja o inserción)?

Justifique su respuesta

R//; Selección.

def seleccion(lista):

    nb = len(lista)

    for actual in range(0,nb):

        mas\_pequeno = actual

        print(lista)

        for j in range(actual+1,nb) :

            if lista[j] < lista[mas\_pequeno] :

                mas\_pequeno = j

        if min is not actual :

            temp = lista[actual]

            lista[actual] = lista[mas\_pequeno]

            lista[mas\_pequeno] = temp

lista = [47,3,21,32,56,92]

seleccion(lista)

print(lista)

Texto

Descripción generada automáticamente

El algoritmo recibe la lista en la primera línea, después de dos pasadas la lista queda como en la línea 3.

1. (10) Utilizar el algoritmo de ordenación SHELL, encuentre las pasadas e intercambios que se realizan para la ordenación de la siguiente lista



def ordenamientoDeShell(unaLista):

    contadorSublistas = len(unaLista)//2

    while contadorSublistas > 0:

      for posicionInicio in range(contadorSublistas):

        brechaOrdenamientoPorInsercion(unaLista,posicionInicio,contadorSublistas)

      print("Después de los incrementos de tamaño",contadorSublistas,

                                   "La lista es",unaLista)

      contadorSublistas = contadorSublistas // 2

def brechaOrdenamientoPorInsercion(unaLista,inicio,brecha):

    for i in range(inicio+brecha,len(unaLista),brecha):

        valorActual = unaLista[i]

        posicion = i

        while posicion>=brecha and unaLista[posicion-brecha]>valorActual:

            unaLista[posicion]=unaLista[posicion-brecha]

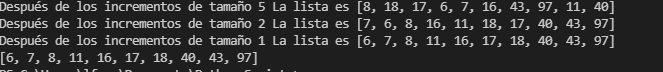
            posicion = posicion-brecha

        unaLista[posicion]=valorActual

unaLista = [8,43,17,6,40,16,18,97,11,7]

ordenamientoDeShell(unaLista)

print(unaLista)



1. (10) Dada una lista voto[0.......n-1], donde cada elemento de lista representa un voto en las elecciones. Suponga que cada voto se da como un número entero que representa el ID del candidato elegido. Desarrolle un algoritmo para determinar quién gana la elección. Determine la complejidad del algoritmo

votos = [1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2]

frecuencia = {}

for i in votos:

if i in frecuencia:

frecuencia[i] += 1

else:

frecuencia[i] = 1

print(frecuencia)

complejidad 3n

1. (10) Se cuenta con una lista de tuplas

futbolistasTup = [(1, "Casillas"), (15, "Ramos"), (3, "Pique"), (5, "Puyol"), (11, "Capdevila"), (14, "Xabi Alonso"), (16, "Busquets"), (8, "Xavi Hernandez"), (18, "Pedrito"), (6, "Iniesta"), (7, "Villa")]

si se aplica futbolistasTup.sort(key=lambda futbolista: futbolista[0])

1. Que resultado se obtiene al aplicar el método .sort

= [(1, "Casillas"), (3, "Pique"), (5, "Puyol"), (6, "Iniesta"), (7, "Villa"),(8, "Xavi Hernandez"), (11, "Capdevila"), (14, "Xabi Alonso"), (16, "Busquets"), (18, "Pedrito").

1. Que se esta especificando en los parámetro (key=lambda futbolista: futbolista[0])

Especifica de que forma se va organizar la lista, por el nombre o el numero.

1. Aplique este método a las listas de los punto 1,3, 4. Que conclusión puede obtener.

Permite hacer diferentes funciones en las respectivas listas, organizarlas con respecto a una especificación(negativos, positivos, decimales, letras, etc).

1. Por favor según opinión realice una tupla con los mejores inventos del 2019 .

Donde usted califica el que mas le gusta o le parece importante.

**Anotación** la escala con la que usted cuenta es de 1 a 100 ( no tiene que asignar ninguno de los extremos si no lo desea)

**Inventos\_2019= [(airPodspro, 65), (eightsleepPod, 70), (roliLumi, 80), (thnkPad X1,70)]**

1. (10) Diseñe e implemente una función para encontrar todos los valores negativos dentro de una lista dada. Tu función debería devolver una nueva lista que contiene los valores negativos.

¿Cuándo ocurre el peor de los casos y cuál es el tiempo de ejecución para ese caso?

import time

lista = [-5,-3,2,1,5,-8,-9]

start = time.time()

negativos = filter(lambda x: x < 0, lista)

lista = list(negativos)

def merge\_sort(lista):

if len(lista) > 1:

mitad = len(lista) // 2

mitad\_izquierda = lista[:mitad]

mitad\_derecha = lista[mitad:]

merge\_sort(mitad\_izquierda)

merge\_sort(mitad\_derecha)

i = 0

j = 0

k = 0

while i < len(mitad\_izquierda) and j < len(mitad\_derecha):

if mitad\_izquierda[i] < mitad\_derecha[j]:

lista[k] = mitad\_izquierda[i]

i = i+1

else:

lista[k] = mitad\_derecha[j]

j = j+1

k = k+1

while i < len(mitad\_izquierda):

lista[k] = mitad\_izquierda[i]

i = i+1

k = k+1

while j < len(mitad\_derecha):

lista[k] = mitad\_derecha[j]

j = j+1

k = k+1

print(lista)

merge\_sort(lista)

print(lista)

end = time.time()

print(end-start)

El peor de los casos es cuando todos los números son negativos.

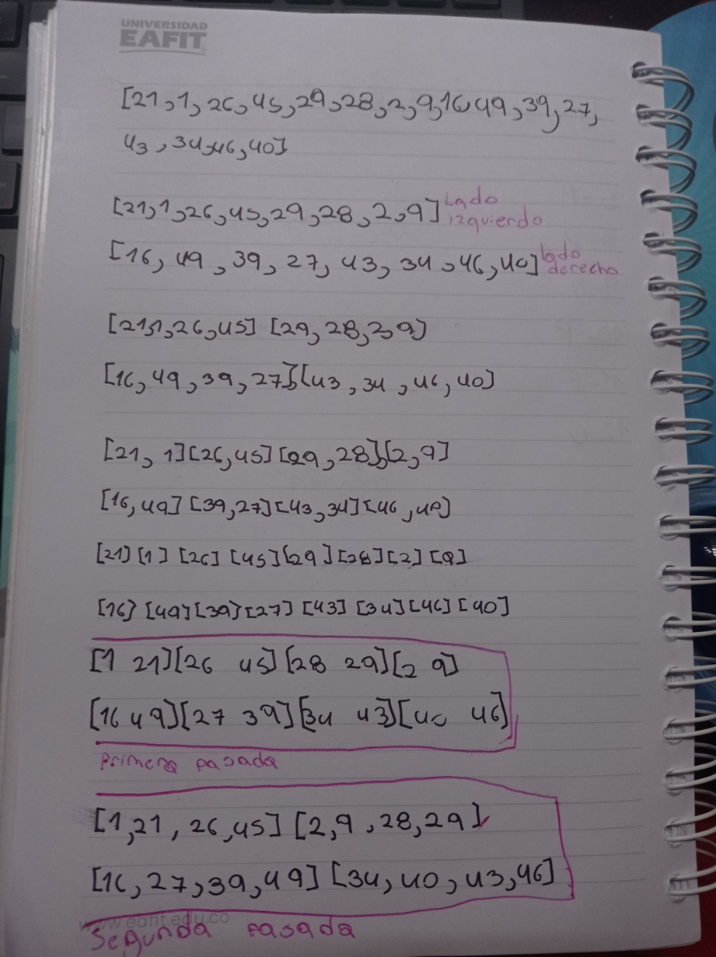
Texto

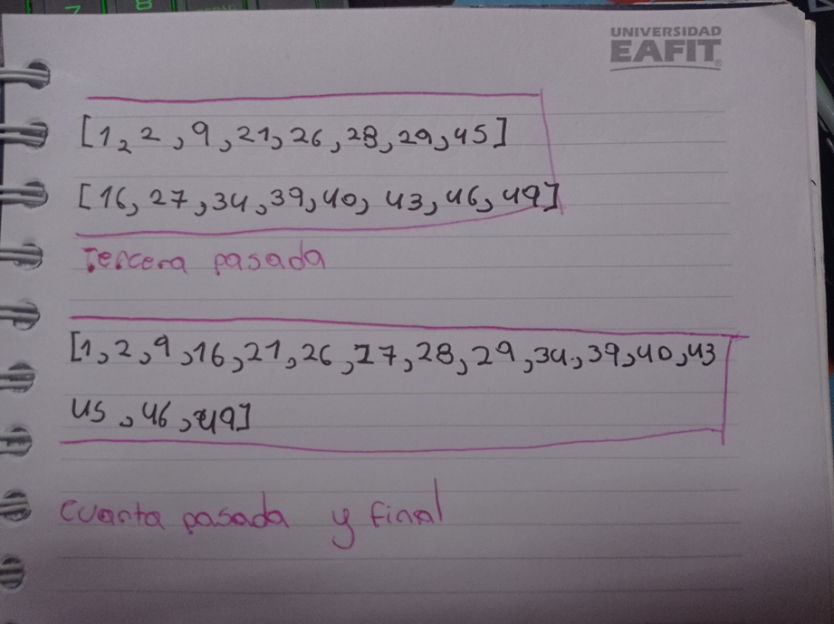
Descripción generada automáticamente

1. (5) Dada la siguiente lista de números:

[21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40]

¿Cuál sería el resultado a la lista que después de 3 llamadas recursivas de ordenamiento por mezcla?





¿Por qué?

La lista se debe separa en arreglos individuales y está compuesta de 16 números, ya con los arreglos individuales se empieza a hacer la comparación en un arreglo de 2 números, luego 4, luego 8 y por último se comparan para el de 16.

El enunciado nos pide parar al tercer llamado, que es cuando quedan dos arreglos de 8 números.

1. (5) Dado el siguiente código evalué su complejidad.

Determine que hace este esté algoritmo

Esplique cada uno de sus funciones y sus clases

* Se crea una clase set, encargada de recibir métodos, funciones y acciones específicas.
* El constructor se encarga de crear la instancia de la clase set.
* Lo siguiente son los métodos.
* Len(self): determina la cantidad de elementos que recibe la clase set.
* Contains(self, element): determina si el elemento se encuentra en la clase o la instancia creada.
* Remove(self, element): elimina elementos enviados, no ingresa a la posición.
* IsSubsetOf(self, setB): determina si una instancia de la clase set es subconjunto de otra.
* Iter(self): crea un iterador que itera los elementos de la clase set.



1. (10) Construya un algoritmo para encontrar un valor específico en una matriz de valores ordenada por filas y columna.

El algoritmo toma como entrada una matriz de valores donde cada fila y cada columna están en orden, junto con un valor para ubicar en esa matriz. Devuelve si ese elemento existe en la matriz.

Por ejemplo, dado la siguiente matriz y buscar el 7, el algoritmo daría como resultado **sí**

Pero si se pide encontrar el número 0, el algoritmo daría como resultado **no**



matris = []

filas = 4

columnas = 6

for i in range (filas):

    matris.append([0]\*columnas)

matris [0][0]=1

matris [0][1]=2

matris [0][2]=2

matris [0][3]=2

matris [0][4]=3

matris [0][5]=4

matris [1][0]=1

matris [1][1]=2

matris [1][2]=3

matris [1][3]=3

matris [1][4]=4

matris [1][5]=5

matris [2][0]=3

matris [2][1]=4

matris [2][2]=4

matris [2][3]=4

matris [2][4]=4

matris [2][5]=6

matris [3][0]=4

matris [3][1]=5

matris [3][2]=6

matris [3][3]=7

matris [3][4]=8

matris [3][5]=9

num\_buscado = int(input())

num = 0

for f in range(filas):

    for c in range(columnas):

        if (matris [f][c] == num\_buscado ):

            print("si")

            num = 1

        elif  (num == 0 and f == 3 and c == 5):

            print ("no")

print (matris)

1. (10) se tiene una lista A con 100 elementos A[ a1……a100 ]

B de 60 elementos B[ b1……b60 ]

Se desean resolver las siguientes tareas

1. Ordenar cada lista aplicando el método Quicksort
2. Crear una lista C que sea la unión de la lista A y B
3. Ordenar la lista C y visualizarla

lista\_a = [1,2,3,4,5,6,7,8,9]

lista\_b = [10,11,12,13,14,15,16,17,18]

def quick\_sort(lista):

    izquierda = []

    derecha = []

    centro=[]

    if len(lista) >1:

        pivote = lista[-1]

        for i in lista:

            if i<pivote:

                izquierda.append(i)

            elif i==pivote:

                    centro.append(i)

            else:

                    derecha.append(i)

        return quick\_sort(izquierda) + centro + quick\_sort(derecha)

    else:

        return lista

def combinar(lista1 , lista2):

    lista\_nueva = []

    tam1=len(lista1)

    tam2=len(lista2)

    if tam1>=tam2 :

        tam3 = tam1

    else:

        tam3 = tam2

    for i in range(0, tam3):

        if i<tam1:

            lista\_nueva.append(lista1[i])

        if i<tam2:

            lista\_nueva.append(lista2[i])

    return lista\_nueva

print(quick\_sort(lista\_a))

print(quick\_sort(lista\_b))

print(quick\_sort(combinar(lista\_a , lista\_b)))

1. (5) Investigue en que consisten las técnicas de ordenación Radixsort y Binsort, en lo posible haga un ejemplo.

**Radixsort:** es un algoritmo de ordenación no comparativa. Este algoritmo evita las comparaciones insertando elementos en cubos de acuerdo con el radix (Radix/Base es el número de dígitos únicos utilizados para representar números. Por ejemplo, los números decimales tienen diez dígitos únicos). Ordena los elementos basándose en los dígitos de los elementos individuales. Realiza la ordenación por conteo de los dígitos desde el menos significativo hasta el más significativo. También se ha llamado ordenación en cubo o digital y es muy útil en máquinas paralelas.

Supongamos que tenemos un array A[] sin ordenar que contiene n elementos.

* [Encuentre el elemento más grande maxm en el array.](https://www.delftstack.com/es/tutorial/algorithm/radix-sort/#encuentre-el-elemento-m%C3%A1s-grande-maxm-en-el-array)
* [Ordena cada dígito de maxm empezando por el menos significativo utilizando un algoritmo de ordenación estable.](https://www.delftstack.com/es/tutorial/algorithm/radix-sort/#ordena-cada-d%C3%ADgito-de-maxm-empezando-por-el-menos-significativo-utilizando-un-algoritmo-de-ordenaci%C3%B3n-estable)

array (1851, 913, 1214, 312, 111, 23, 41, 9)



**Binsort:** es un [algoritmo de ordenamiento](https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento) que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero solo puede contener los elementos que cumplan unas determinadas condiciones. En el ejemplo esas condiciones son intervalos de números. Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos.

Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo para obtener casilleros con menos elementos.

El algoritmo contiene los siguientes pasos:

1. Crear una colección de casilleros vacíos.
2. Colocar cada elemento a ordenar en un único casillero.
3. Ordenar individualmente cada casillero.
4. devolver los elementos de cada casillero concatenados por orden.

