Algoritmos de ordenamiento

Grupo 7

Integrantes: Enzo Candio, Tiago Schmidt, Christian Burlone

Este trabajo práctico grupal tiene como objetivo investigar y desarrollar distintos tipos de algoritmos de organización, desarrollar su código en Python y poder explicar su funcionamiento además de su aplicación en la situación propuesta por los docentes. Con este objetivo desarrollamos los siguientes algoritmos de organización: Bubble Sort, Cocktail Sort, Insertion Sort y Selection. Sort. Los mismos fueron desarrollados por cada miembro del grupo.

Breve explicación:

Cocktail Sort: Este algoritmo es considerado una versión mejorada de “Bubble sort” ya que recorre la lista ida y vuelta (izquierda-derecha|derecha-izquierda). Para esto hay que programar dicho algoritmo para que realice un swap y lleve los números mayores a la derecha y los menores a la izquierda, reduciendo así las puntas.

BubbleSort: El sistema de organización de burbuja o Bubblesort se encarga de organizar los elementos asignados mediante una comparación de pares de forma adyacente. Por ejemplo, si tenemos una lista de números variados y se nos pide que se organice de forma que los más pequeños queden a la izquierda y los más grandes a la derecha, lo que hará el programa es:

- 1-) Dada una lista [10,50,15,30,25]

  El programa preguntará: ¿el primer número es más grande que el segundo? Si fuese el caso de que el número de la izquierda es más grande que el de la derecha, se ejecuta un swap: en el caso contrario, no se ejecuta el swap. En nuestro caso, con la lista de números, no se ejecuta un swap ya que el primer par de números, que es 10 y 50, muestra que 10 no es más grande que 50.

- 2-) [10,15,50,30,25]

  En la segunda iteración vemos que sí se ejecuta un swap: el 50 sí es más grande que el 15. Entonces, siguiendo la misma lógica, el programa se preguntará lo mismo hasta que el número más grande quede al final.

- 3-) [10,15,30,25,50]

Una vez que llegamos a esta situación, donde el número más grande ya quedó al final, el programa comenzará a ordenar nuevamente los números de a pares, sin incluir al 50. Cuando termina esta pasada, la lista debería quedar así:

- 4-) [10,15,25,30,50] (reestructurar el formato de esta sección)

Insertion Sort: Algoritmo de organización por inserción: El sistema de organización por inserción toma un elemento de la lista y se va comparando hacia la **izquierda**, Mientras que el elemento que lleva a cabo el recorrido sea menor que los elementos ordenados previamente, continúa su recorrido y una vez encontrada su posición correcta se inserta en su lugar correspondiente. Para que luego el algoritmo tome el siguiente elemento y repita el ciclo hasta ordenar la lista.

- Un ejemplo de este algoritmo utilizando lista de números podría ser:

  [5] | [4,3,7,10]

- Vemos que tenemos 2 listas, lo que queremos hacer es que ambas listes se unifiquen y queden ordenadas de menor a mayor, para eso usamos el Insertion Sort y quedaría de la siguiente forma

  [4,5] | [3,7,10]

- Vemos que el número 4 se insertó a la izquierda del número 5 esto es ya que el número 4 recorrido la lista hacia la izquierda encontrando que numero 5 es más grande y como el número que recorre es más pequeño se inserta a la derecha

  [3,4,5] | [7,10]

- En el caso con el 3 pasa exactamente lo mismo al recorrer desde la izquierda se compara con todos los números hasta encontrar su lugar e insertarse

  [3,4,5,7] | [10]

- El 7 solo hace 1 iteración ya que al comprar con el número 5 y ver que es mas alto que el se coloca por a la derecha, lo mismo pasará con el 10 se compara con el 7 y como sabe que el 7 es el último número y que el 10 es mayor se coloca a la derecha de 7 (reestructurar el formato de esta sección)

Selection Sort: Es uno de los algoritmos de inserción más simples, itera a través de la lista seleccionando el elemento más pequeño e intercambiandolo para ubicarlo en la posición correcta, funciona de la siguiente manera:

1 - Encuentra el elemento más pequeño de la lista y lo intercambia con la primer posición.

2 - Encuentra el segundo elemento más pequeño y lo intercambia con la segunda posición.

Repite el proceso de encontrar el siguiente elemento más pequeño y swapearlo en la posición correcta hasta que se ordene toda la lista.

Códigos de los programas:

BubbleSort:

```
items = []
n = 0
i = 0
j = 0

def init(vals):
    global items, n, i, j
    items = list(vals)
    n = len(items)
    i = 0
    j = 0

def step():
    global items, n, i, j, swapped
```

```
    if i>=n-1:
        return {"a": -1, "b": -1, "swap": False, "done": True}
    if j<n-i-1:
        if items[j]>items[j+1]:
            items[j], items[j+1]=items[j+1],items[j]
            swapped=True
            resultado= {"a":j, "b":j+1 ,"swap":True, "done":False}
        else:
            resultado= {"a":j, "b":j+1 ,"swap":False, "done":False}
        j=j+1
        return resultado
    else:
        if not swapped:
```

InsertionSort:

```
items = []
n = 0
i = 0
j = None

def init(vals):
            global items, n, i, j
            items = list(vals)
            n = len(items)
            i = 1
            j = None

def step():
            global items, n, i, j
```

```
            if i>=n:
            return{"a": -1, "b": i-1, "swap":False, "done": True}
            if j ==None:
            j=i
            return{"a":i, "b":i, "swap":False, "done": False}
            if j>0 and items[j-1] > items[j]:
            items[j-1], items[j] = items[j], items[j-1]
            ret={"a": j-1, "b": j, "swap": True, "done": False}
            j=j-1
            return ret
            else:
            i=i+1
            j=None
            return {"a": -1, "b": i-1, "swap": False, "done": False}
```

SelectionSort:

```
def step():
    global items, n, i, j, min_idx, fase

    if i >= n-1:
        return {"done": True}

    if fase == "buscar":
        if items[j] < items[min_idx]:
            min_idx = j
        j += 1

        if j >= n:
            fase = "swap"
```

```
            return {"a": min_idx, "b": j-1, "swap": False, "done": False}

        return {"a": min_idx, "b": j, "swap": False, "done": False}

    elif fase == "swap":
        swap = False
        a = i
        b = min_idx

        if min_idx != i:
            items[i], items[min_idx] = items[min_idx], items[i]
            swap = True

        i += 1
        j = i + 1
        min_idx = i
        fase = "buscar"

        if i >= n-1:
            return {"done": True}

        return {"a": a, "b": b, "swap": swap, "done": False}
```

CocktailSort:

```
def init(vals):
    global items, n, j, izq, der, tecla
    items = list(vals)
    n = len(items)
    j = 0
```

```
    izq = 0
    der = n - 1
    tecla = True

def step():
    global items, n, j, izq, der, tecla

    if izq >= der:
        return {"done": True}

    swap = False

    if tecla:
        a = j
        b = j + 1

        if items[a] > items[b]:
            items[a], items[b] = items[b], items[a]
            swap = True
        j += 1

        if j >= der:
            tecla = False
            der -= 1

    else:
        a = j
        b = j - 1
```

```
        if items[a] < items[b]:
            items[a], items[b] = items[b], items[a]
            swap = True
        j -= 1

        if j<=izq:
            tecla = True
            izq += 1

    return {"a": a, "b": b, "swap": swap, "done": False}
```

Dificultades:

Tiago: Algunas de las dificultades que tuve al realizar el trabajo fue entender la lógica del código de los sistemas de organización si bien entendía su funcionamiento se complicó bastante a la hora de pasar la idea a código, me ayudo bastante algunos videos de YouTube que explican el código y su lógica además de algunas páginas. Me agarré mucho de unos videos específicos de un mismo creador de contenido para realizar mi parte del trabajo. otra complicación que tuve fue que muchas veces los programas no funcionaban y tenía que identificar el error, o se ordenaban de otra forma que no era la propuesta, muchas veces quizás el programa andaba, pero por no amigarme mucho con el funcionamiento de github y git bash se me complicaba poder hacer pruebas.

Christian: El principal desafío que encontré durante el desarrollo del código fue la implementación de la teoría de los algoritmos. Aunque pude comprender su funcionamiento, la estructuración de variables y la traducción de la lógica al código funcional resultó ser una tarea un poco compleja.

Antes de programar, me concentré en entender la mecánica de los algoritmos, lo cual, aunque teóricamente sencillo, presentó dificultades al intentar trasladar mi interpretación a un código operativo. Después de múltiples intentos, llegué a una versión funcional. Posteriormente, recurrí a la asistencia de una IA para optimizar el código y detectar posibles fallos. Tras varias correcciones, alcancé la versión final entregada, a la cual también le añadí anotaciones más claras y precisas.

Nota: El apartado de “métricas” y su implementación la hicimos en su totalidad con Gemini y Copilot

Bibliografía:

-https://www.youtube.com/watch?v=pqZ04TT15PQ&t=118s

-https://ellibrodepython.com/bubble-sort

-https://www.youtube.com/watch?v=6GU6AGEWYJY

-https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/

-https://github.com/gbaudino/MetodosDeOrdenamiento

-https://github.com/TheAlgorithms/Python/blob/master/divide_and_conquer%2Fmergesort.py