INFORME

TRABAJO PRÁCTICO N°1

“Algoritmos de ordenamiento”

Comisión-5 Semestre 2.2025

DOCENTES: PATRICIA BAGNES Y FERNANDO VELCIC

GRUPO: CAMILA BENITEZ MARIN

Resumen: El trabajo práctico consiste en la implementación de algoritmos de ordenamiento y la visualización de los mismos desde el navegador

INTRODUCCIÓN:

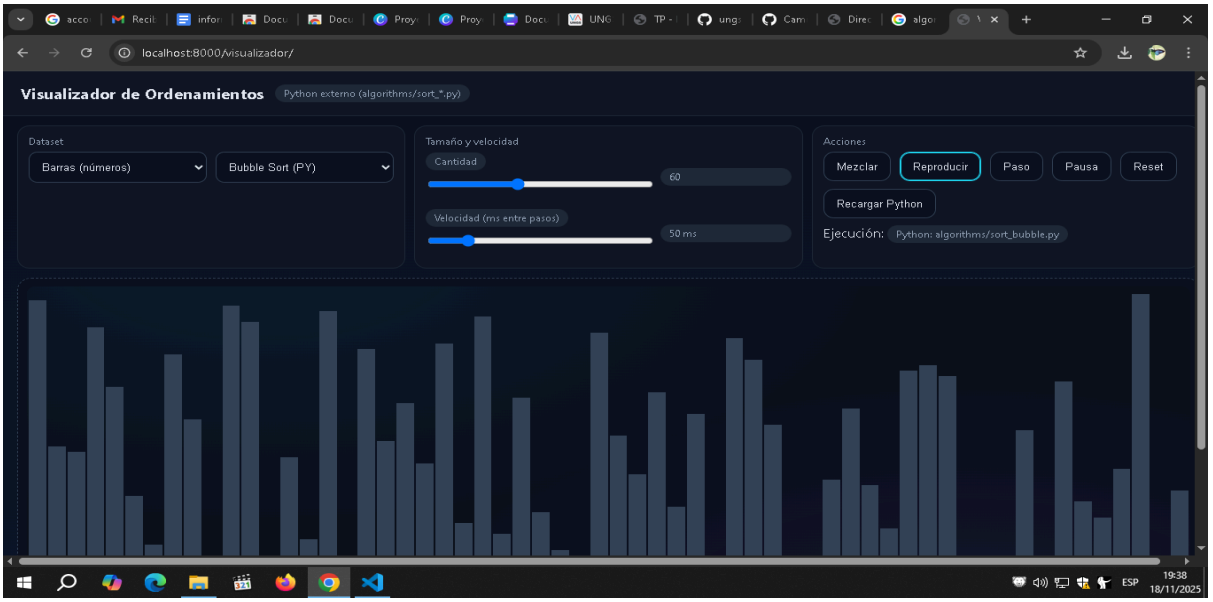
El trabajo práctico consiste en la implementación de **algoritmos de ordenamiento** y la visualización de los mismos desde el navegador . **Los algoritmos de ordenamiento** son un conjunto de instrucciones que recibe como entrada una lista para organizar sus elementos en una secuencia específica, como orden ascendente o descendente, numérico o alfabético.

En este trabajo se completó el código del visualizador con tres algoritmos: **Bubble sort, Selection sort, Insertion sort,** cumpliendo con el contrato que utiliza el visualizador. Bajo un modelo de ejecución paso a paso. Cada llamada a la función step() debe realizar la unidad mínima de trabajo (una comparación o un intercambio). Esto requiere utilizar variables globales para continuar el estado del algoritmo entre llamadas, simulando los bucles for tradicionales.

El Contrato de la Función step()

La función step() debe retornar un diccionario con la siguiente información:
a (int): Índice del primer elemento involucrado (comparado o intercambiado).
b (int): Índice del segundo elemento involucrado (comparado o intercambiado).
swap (bool): Es True si se realizó un intercambio de datos. Es False si solo fue una comparación.
done (bool): Es True si el algoritmo ha finalizado el ordenamiento.

El visualizador:



¿Què funciòn realiza cada “botòn”?

Reproducir: llama a step() en bucle con pausas, este permite que que las barras del visualizador se intercambien automáticamente, sin realizar los pasos manualmente.
Paso: llama a step() una sola vez**,** esto permite observar cómo se realiza el intercambio de menor a mayor paso a paso.

Mezclar: Desordena la lista para volver a ordenarla con “reproducir” o “paso”.

Reset: Una vez presionado “reproducir” o “paso” reset reinicia los intercambios, sin mezclar la lista:

La consigna:

La consigna del trabajo práctico pide implementar algoritmos de ordenamiento utilizando python, donde se debe completar el código cumpliendo el contrato init(vals) + step() que usa el visualizador. Este se encuentra alojado en un repositorio en Github, el cual debe copiarse(“fork”) para poder completarlo.

Bubble Sort (Ordenamiento Burbuja)

```
15  def step():
16      #TODO:
17      global items, n, i, j
18      # 1) Elegir índices a y b a comparar en este micro-paso (según tu Bubble).
19      a=j
20      b=j+1 #adyacente del indice,elemento de al lado
21   # 2) Si corresponde, hacer el intercambio real en items[a], items[b] y marcar swap=True
22      if  i < n-1:
23        if items[a]>items[b]:#sin ciclos for porque funciona "paso a paso"
24            items[a],items[b]= items[b],items[a] #intercambio de menor a mayof
25            swap=True
26        else:
27            swap=False
28
29      j=j+1#aumento el indice
30
31      if j >= n-i-1:#si hay que reiniciar el indice
32          j=0
33          i=i+1
34        #Devolver {"a": a, "b": b, "swap": swap, "done": False}.
35      return  {"a": a, "b": b, "swap": swap, "done": False}
36      # Cuando no queden pasos, devolver {"done": True}.
37      if i >=n-1:
38          return {"done": True}
```

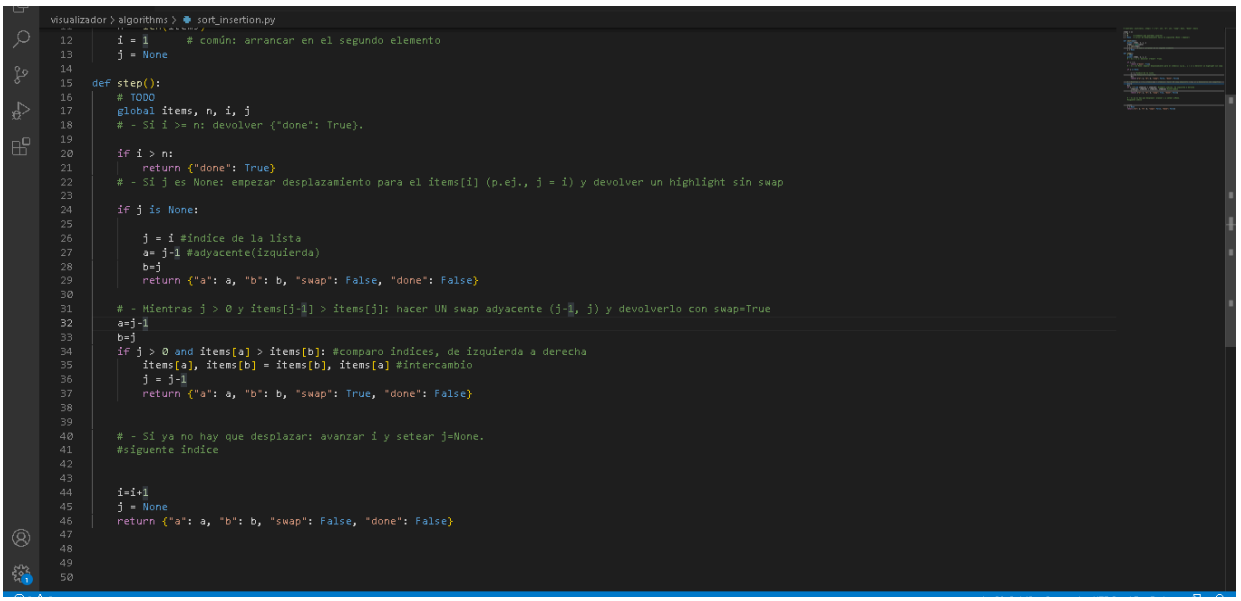
El **Bubble Sort** compara y potencialmente intercambia elementos adyacentes, haciendo que el elemento más grande "burbujee" hasta su posición final en cada pasada.
Variables de Estado Globales:

i: Contador del bucle externo. Rastrea cuántos elementos ya están ordenados al final de la lista.

j: Cursor del bucle interno. Recorre la porción no ordenada comparando adyacentes (a=j y b=j+1)
swap: Bandera que indica si hubo algún intercambio en la pasada actual. Se usa para la optimización: si no hay swaps, el algoritmo termina.

La dificultad principal con este algoritmo fue a la hora de completarlo, ya que a pesar de tener ejemplos de internet, teniamos que adaptarlo y no podíamos usar los ciclos For, porque había que hacerlo “paso a paso" .

Insertion Sort (Ordenamiento por Inserción)



El **Insertion Sort** toma el elemento en la posición i y lo inserta en su lugar correcto dentro de la sublista ordenada a su izquierda, desplazando los elementos mayores.
Variables de Estado Globales:

i: Puntero del bucle externo. Marca el elemento que se va a insertar.
j: Puntero del bucle interno. Cursor que se mueve hacia la izquierda (j-1 y j) para desplazar elementos y encontrar la posición de inserción.

La dificultad principal al implementar este algoritmo fue a la hora de actualizar correctamente los índices, especialmente el índice i, a y b(estos índices hacen que avance la comparación de las barras en el visualizador). Estos problemas en el visualizador hacían que no se recorran completamente las barras y las ordene.

Selection Sort (Ordenamiento por Selección)

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

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

    if fase == "buscar":

        if j < n:
            a = min_idx
            b = j

            if items[j] < items[min_idx]:
                min_idx = j
                a = min_idx

            j += 1

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

        else:
            fase = "swap"
            return {"a": i, "b": min_idx, "swap": False, "done": False}

    elif fase == "swap":

        swap_realizado = False
        a = i
        b = min_idx

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

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

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

El **Selection Sort** encuentra el elemento mínimo en la porción no ordenada (desde i) y lo intercambia con el elemento en la posición i. Utiliza la variable fase para controlar la ejecución.

Variables de Estado Globales:

i: Puntero del bucle externo. Marca el inicio de la porción no ordenada y la posición final del elemento mínimo.
j: Puntero del bucle interno. Cursor que recorre la porción no ordenada buscando el mínimo.
min_idx: Guarda el índice del valor mínimo encontrado en la pasada actual.
fase: Controla el flujo: "buscar" (encuentra el mínimo) o "swap" (realiza el intercambio y reinicia).

La principal dificultad fue al implementar este algoritmo en el formato "paso a paso" sin usar ciclos for, fue en la coordinación precisa entre las dos fases del algoritmo y el reinicio de los índices.

Concluciòn:

Con este trabajo práctico, además de aprender cómo funciona cada uno de los tres algoritmos aplicados, también pude interpretar otros códigos al momento de investigar, y llegar a modificarlos según las necesidades del trabajo práctico.

Una de las principales dificultades fue entender lo que me pedían hacer. Mi primer error fue empezar a completar el trabajo sin antes leer detenidamente el contrato y las características de cómo funcionaba el visualizador. Esto me llevó a cometer algunos errores que pude corregir una vez que comprendí mejor los requerimientos.

Fue interesante investigar como diferentes algoritmos resuelven el mismo problema, pero tenían una forma diferente de hacerlo y a la vez una duración diferente.