Brezovsky, Tomás

Dificultades:
-Uso de documentación que se encuentra disponible en inglés, ya que mi nivel de inglés es básico. Tuve que optar por el uso de traductores.
-Modificación de *index.html* para agregar el algoritmo gnome sort en el menú desplegable.

Python docs - https://docs.python.org/3.12/tutorial/datastructures.html
GeeksForGeeks: Bubble Sort (definición y pseudocódigo) - https://www.geeksforgeeks.org/dsa/bubble-sort-algorithm/
Learn Bubble Sort in 7 minutes - https://www.youtube.com/watch?v=Dv4qLJcxus8
Gnome Sort: definición y pseudocódigo (GeeksForGeeks) - https://www.geeksforgeeks.org/dsa/gnome-sort-a-stupid-one/
Gnome Sort: Wikipedia (origen y comportamiento) - https://es.wikipedia.org/wiki/Gnome_sort

Uso de bubble sort: sort_bubble.py

Opté por el uso de este algoritmo por la facilidad de su codificación. La sencillez de su escritura reside en que mediante él se busca un orden de menor a mayor, y lo logra comparando elementos vecinos e intercambiándose si están al revés. El algoritmo revisa, compara y, si es necesario, intercambia continuado el desplazamiento hacia la derecha.

```
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

   #Si ya terminó o no hay nada que comparar.
   if n<=1 or i>=n-1:
       return {"done":True}
```

```
    #Si j llegó al límite de comparación para esta pasada.
    if j>=n-i-1:
        j=0
        i+=1

        #Si después de incrementar i ya está ordenado.
        if i>=n-1:
            return {"done":True}

    #Compara los elementos adyacentes.
    a=j
    b=j+1
    swap=False

    #Hace el swap real antes de devolverlo.
    if items[a]>items[b]:
       items[a],items[b]=items[b],items[a]
       swap=True

    #Se avanza a j para el proximo micro-paso.
    j+=1

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

Uso de gnome sort: sort_gnome.py (opcional)

Para implementar Gnome Sort, la dificultad estuvo en incluirlo en el menú desplegable de la UI. Ingresé en *index.html* y busqué otro algoritmo de ordenamiento ya implementado (como bubble sort). Hice la prueba de copiar el mismo código del llamado al otro algoritmo y agregar una nueva línea en *index.html*, pero en ese caso con la modificación del nombre del algoritmo. Quedó implementado en la línea 401: *<option value="gnome">Gnome Sort (PY)</option>*

```
items = []
n = 0
# Agregá acá tus punteros/estado, p.ej.:
# i = 0; j = 0; fase = "x"; stack = []

#Estado del Gnome Sort
pos=0
```

```
def init(vals):
   global items,n,pos
   items = list(vals)
   n = len(items)

def step():
   global items,n,pos

   #Si ya terminó.
   if n<=1 or pos>=n:
       return {"done":True}

   #Si estamos al principio, solo avanzamos.
   if pos==0:
       pos+=1
       return{"a":0,"b":0,"swap":False,"done":False}

   a=pos-1
   b=pos

   #Caso 1: desorden -> swap y retrocedemos.
   if items[a]>items[b]:
       items[a],items[b] = items[b],items[a]
       pos-=1
       return {"a":a,"b":b,"swap":True,"done":False}

   # Caso 2: orden -> avanzamos.
   else:
       pos+=1
       return {"a":a,"b":b,"swap":False,"done":False}
```

Marrero, Ignacio

Dificultades:
-Al igual que mi compañero, mi nivel de inglés es básico, por ende el uso de documentación que se encuentra disponible en ese idioma fue uno de los obstáculos que tuve que sortear con aplicaciones de traducción.
-El uso de json me resultó complicado, investigando al respecto descubrí que se puede hacer validaciones (aplicar condicionales) dentro del bloque de código lo que me facilitó la implementación en el *insertion sort* y pude evitar la repetición de código y el uso de if´s anidados.

Python docs - https://docs.python.org/3.12/tutorial/datastructures.html
Insertion sort - https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n
Selection sort - https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n
Json - https://www.json.org/json-es.html

Uso de insertion sort: sort_insertion.py

La definición de este algoritmo me resultó amigable. Su funcionamiento se basa en la comparación de un valor con su vecino anterior y si este es mayor invierte su posición hasta encontrar a su vecino anterior de menor valor.

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


    # Si ya terminamos
    if i >= n:
        return {"done": True}


    # Si j es None, empezar a desplazar
    if j is None:
        j = i
        return {
            "a": j - 1 if j > 0 else 0,
            "b": j,
            "swap": False,
            "done": False
        }
```

```
    # Si todavía hay desplazamiento (comparar y swapear)
    if j > 0 and items[j - 1] > items[j]:

        items[j - 1], items[j] = items[j], items[j - 1]
        j -= 1

        return {
            "a": j,
            "b": j + 1,
            "swap": True,
            "done": False
        }

    # Si no hay más desplazamiento avanzar al siguiente i
    i += 1
    j = None
    return {
        "a": i - 1,
        "b": i,
        "swap": False,
        "done": False
    }
```

Uso de selection sort: sort_selection.py

Este algoritmo cuenta con dos fases, en primer lugar realiza una búsqueda global entre todos los valores y selecciona el mínimo.
Luego, entra en la fase de intercambio de valores y realiza una comparación con la posición del mínimo encontrado y la ubicación en la que partió la búsqueda, si estos son distintos invierte los valores y avanza con una nueva fase de búsqueda desde la posición siguiente.
La mayor dificultad aquí fue detectar en qué momento se deben implementar los cambios de fase.

```
    global items, n, i, j, min_idx, fase


    # Si i llegó a n-1, ya no quedan pasadas
    if i >= n - 1:
        return {"done": True}
```

```
    # FASE 1: BUSCAR MÍNIMO

    if fase == "buscar":
        # Si j está dentro del rango, seguimos comparando
        if j < n:
            # Comparar el elemento actual con el mínimo encontrado
            actual_j = j
            if items[actual_j] < items[min_idx]:
                min_idx = actual_j

            # Avanzar j para la siguiente llamada
            j += 1

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

        # Si j salió del rango terminamos la fase de búsqueda
        fase = "swap"

    # FASE 2: HACER EL SWAP

    if fase == "swap":

        # Si el mínimo no está en i, hacemos el swap
        if min_idx != i:
            items[i], items[min_idx] = items[min_idx], items[i]

            # Después del swap se avanza i y se reinician valores, pero
devolvemos primero el swap visual
            anterior_i = i
            anterior_min = min_idx

            i += 1
            j = i + 1
            min_idx = i
```

```
            fase = "buscar"

            return {
                "a": anterior_i,
                "b": anterior_min,
                "swap": True,
                "done": False
            }

        # Si no hay swap, avanzar i
        i += 1
        j = i + 1
        min_idx = i
        fase = "buscar"
        return {
            "a": i - 1,
            "b": min_idx,
            "swap": False,
            "done": False
        }
```