TP – Visualización de algoritmos de ordenamiento

Introducción (¿qué es un algoritmo de ordenamiento?)

Un **algoritmo de ordenamiento** es un procedimiento que re-acomoda una colección de elementos según un criterio, por ejemplo:

- ordenar números de menor a mayor,
- ordenar palabras alfabéticamente,
- ordenar objetos por alguna propiedad (precio, fecha, etc.).

Existen muchas estrategias (Bubble, Selection, Insertion, Quick, Merge…), cada una con una idea distinta para comparar e intercambiar elementos.

¿Qué haremos y cómo se conectará con Python?

Vas a implementar algoritmos de ordenamiento **en Python**, y ver su ejecución **animada en el navegador**. Esto se alinea con lo visto en la materia:

- **Variables** para guardar estado.
- **Listas** e **índices** para acceder e intercambiar elementos.
- **Condicionales** para decidir intercambios.
- **Bucles** modelados como **punteros** que avanzan entre llamadas (en lugar de `for` grandes, el TP usa la idea de “un paso por vez”).
- **Booleanos** para indicar acciones (por ejemplo, si hubo intercambio).

Objetivo

Implementar **al menos 3 algoritmos distintos** de ordenamiento cumpliendo el contrato `init(vals)` + `step()` que usa el visualizador.

A implementar:

- **Bubble**
- **Selection**
- **Insertion**.

Opcionales / extra (para quienes quieran sumar puntos):

- **Quick** (iterativo con pila), **Merge** (bottom-up), **Shell**, **Heap**, etc.
- **Métricas** (contadores de comparaciones/intercambios) y análisis breve.

Estructura del proyecto y ejecución

```
/visualizador/
  index.html              # visualizador web (provisto)
  /algorithms/
    sort_bubble.py
    sort_selection.py
    sort_insertion.py
    sort_template.py        # plantilla para agregar nuevos
```

podés sumar sort_quick.py, sort_merge.py, sort_shell.py, ...

Cómo correr:

En `/visualizador` ejecutar:

1. **python -m http.server**
2. Abrir `http://localhost:8000` (ideal en incógnito).
3. Elegir **dataset** y **algoritmo** en los selectores.
4. Botones: **Mezclar**, **Reproducir**, **Paso**, **Pausa**, **Reset**.

> El selector de algoritmo carga automáticamente tu archivo `algorithms/sort_<valor>.py`.

Dónde programar y qué tocar

- Todo tu código va en **`/algorithms/`**.
- Para crear un algoritmo nuevo, copiá `sort_template.py` → `sort_mialgo.py` y **agregá** la opción en el `<select id="algorithm">` del `index.html` con `value="mialgo"`.
- **No es necesario modificar `index.html`** para los algoritmos provistos.

Contrato que debe cumplir tu código

Cada archivo `sort_<algo>.py` debe exponer **dos funciones** globales:

`init(vals: list[int]) -> None`

Se llama **una vez** al comenzar (o cuando se remezcla). Debés:

- Guardar una **copia**: `items = list(vals)`.
- Guardar `n = len(items)`.
- Inicializar el **estado/punteros** de tu algoritmo (p. ej. `i`, `j`, `min_idx`, una pila, etc.).

`step() -> dict`

Se llama **muchas veces**. Cada llamada realiza **UN solo micro-paso** y devuelve:

```
{
  "a": int,    # índice A para resaltar (0..n-1)
  "b": int,    # índice B para resaltar (0..n-1)
  "swap": bool,  # True si en ESTE paso intercambiaste items[a] <-> items[b]
  "done": bool   # True cuando el algoritmo terminó
}
```

Sobre `swap`:

- `swap=True` le indica al visualizador que **también** mueva las columnas A y B.

Vos **debés realizar el mismo swap** en tu lista Python **antes** de devolver el dict:

```
 items[a], items[b] = items[b], items[a]
return {"a": a, "b": b, "swap": True, "done": False}
```

-

Siempre debes cumplir:

- `0 <= a,b < n` siempre que devuelvas índices.
- Si `swap=True`, ya hiciste ese intercambio en `items`.
- Al finalizar, devolves `{"done": True}` y detener la secuencia de pasos.
- Actualizar correctamente tus punteros/estado en cada paso.

Cómo usa la UI tu `step()`

- **Reproducir**: llama a `step()` en bucle con pausas.
- **Paso**: llama a `step()` **una sola vez**.
- Si `step()` devuelve `{"done": True}`, la ejecución se detiene y se limpia el resaltado.
- Si `swap=True`, se destaca `(a,b)` y luego el visualizador intercambia esos dos elementos en pantalla para sincronizar con tu lista.

Extensión de imagen por columnas (¿deben implementarla?)

- **No.** La extensión de “Imagen por columnas” ya está **provista** en el visualizador y **no requiere cambios de tu parte**. Tus algoritmos trabajan sobre una lista de enteros y la UI se encarga de mostrar/mover las columnas.
- Implementar desde cero el “corte de imagen” implicaría manipulación de imágenes en el navegador (JavaScript/Canvas) y **queda fuera de alcance** de esta materia..

Entregable

- **(Obligatorio)** Carpeta `/algorithms/` con **al menos 3** archivos funcionando (ej.: `sort_bubble.py`, `sort_selection.py`, `sort_insertion.py`).
- **(Obligatorio)** Informe detallado documentando los algoritmos implementados junto con las decisiones tomadas, implementaciones aplicadas y dificultades encontradas.
- **(Obligatorio)** Un **README.md** breve con: integrantes, algoritmos implementados y una nota corta sobre decisiones de implementación.
- **(Opcional)** Algoritmos extra (`sort_quick.py`, `sort_merge.py`, `sort_shell.py`, …)
- **(Opcional)** Agregar métricas sobre cantidad swaps, tiempo de duración, etc
- **(opcional)** Quien quiera explorar el código JS para cambiar cómo se corta/mezcla la imagen puede hacerlo y documentarlo como anexo.

Errores comunes (y cómo evitarlos)

- `swap=True` **sin swap real**: la animación se desincroniza. Hacé primero el intercambio en `items` y luego devolvé `swap=True` con esos índices.

- **Índices fuera de rango**: cuidá límites (ej. en Bubble, `j+1 < n-i`).

- **Nunca llega `done`**: asegurá los avances de fase/anchura/pila al completar una pasada/bloque.

- **No reseteás estado en `init`**: remezclás y queda estado viejo; siempre reinicializá.