

## Informe – Algoritmos Bubble Sort, Insertion Sort y Selection Sort

En este trabajo desarrollamos tres algoritmos clásicos de ordenamiento (Bubble Sort, Insertion Sort y Selection Sort, todos adaptados para ejecutarse paso a paso según el contrato `init / step`. Aunque el objetivo general de los tres es ordenar una lista comparando e intercambiando elementos, cada algoritmo requiere una adaptación específica para que la visualización pueda mostrar exactamente lo que ocurre en cada llamada a `step`. Eso hizo que el desafío fuera tanto lógico como de diseño: no bastaba con que el algoritmo funcionara, había que descomponerlo en microacciones y respetar estrictamente el formato que pide la interfaz.

En el caso de **Bubble Sort**, una de las principales dificultades fue manejar bien los índices `i` y `j` para avanzar comparación por comparación, sin saltar pares y sin pasarnos de los límites de cada pasada. Si bien el algoritmo en sí es simple, traducirlo a una ejecución paso a paso requirió tener mucho control sobre cada movimiento. También tuvimos que asegurarnos de que los intercambios se realizaran antes de devolver el diccionario con la información del paso, tal como lo pedía el contrato del proyecto. Ese detalle era importante porque afectaba directamente a cómo se mostraban los swaps en la visualización. Además, otro desafío fue integrar el código dentro del repositorio del proyecto: la estructura inicial no nos resultó del todo intuitiva y hubo que probar varias ubicaciones y ajustes hasta que la app leyó correctamente los archivos. Después de varias pruebas y revisiones dentro del equipo, logramos que el Bubble Sort funcionara sin errores en la interfaz.

Por otro lado, implementamos **Insertion Sort** siguiendo la misma metodología paso a paso. En este caso, la dificultad estuvo en entender cómo manejar el índice principal `i` y el cursor `j`, que comienza como `None` para indicar que empieza una nueva iteración; `None` no significaba un error, sino una señal para que en la siguiente llamada `j` se inicializara en `i`. Otro punto que al principio costó fue interpretar por qué, después de mover `j` hacia la izquierda con `j = j - 1`, la función debía devolver "a": `j` y "b": `j+1`: tras analizar cómo lo interpreta la visualización, entendimos que esos valores representan los dos elementos que acababan de compararse o intercambiarse, y por eso hay que actualizarlos después del decremento. También aprendimos a utilizar la palabra clave `global` para conservar los valores de `items`, `i`, `j` y `n` entre llamadas; aunque es una herramienta sencilla, resultó necesaria para mantener el estado entre pasos. En síntesis, en Insertion Sort lo más complicado no fue la idea del algoritmo, sino descomponerla en microacciones y coordinar índices para que cada `step()` realizara exactamente una acción reconocible por la animación.

Adicionalmente, implementamos **Selection Sort** de forma incremental para ajustarlo al mismo formato. La idea general que seguimos fue recorrer la parte no ordenada buscando el menor `y`, si corresponde, hacer un swap al final de la pasada. Para que esto funcione paso a paso, se dividió el proceso en fases: una fase de búsqueda (donde se comparan elementos y se actualiza el candidato a mínimo) y una fase de swap (donde se realiza el intercambio único con la cabeza de la porción no ordenada). Al principio nos costó interpretar la consigna y definir cómo avanzar exactamente entre comparaciones, por eso decidimos usar estado global que permita recordar `i`, `j` y `min_idx` entre llamadas. También se optó por separar claramente las fases para que la ejecución sea más fácil de seguir y para evitar actualizaciones erróneas de índices. Hubo que manejar con cuidado los índices para evitar comparaciones fuera de rango y

para que el swap se produjera sólo cuando correspondía; además fue importante devolver en cada paso los datos que la interfaz espera para que la visualización muestre correctamente la comparación o el intercambio.

En síntesis, entre los tres algoritmos los desafíos principales fueron coordinar el control de índices, organizar la ejecución paso a paso sin romper el contrato del simulador, integrar correctamente el código al repositorio y ponerse de acuerdo en pequeños detalles de implementación (por ejemplo, cuándo exactamente devolver cada tipo de retorno). Con paciencia, pruebas y colaboración entre los tres integrantes del grupo, pudimos completar las implementaciones y dejar cada algoritmo funcionando en la visualización como pedía la consigna.