



# Metodos de Ordenamiento: Selection sort

**Nombre del alumnos:**

**Sergio Alexander Antonio Gracida**

**Luis Alberto Figueroa gonzalez**

**Carlos Antonio Cortes Torres**

**Arath Yahir Lopez Guzman**

**Oswaldo Martínez vidaña**

**Angel David Garcia Blas**

**Profesor: Kevin David Molina Gómez**

**Materia: Estructura de Datos**

**Fecha: 02/08/2025**

# Selection\_Sort.py

## Explicación:

```
def metodo_seleccion(arreglo):
```

Esta función se llama `metodo_seleccion`, y su propósito principal es ordenar un arreglo utilizando el clásico algoritmo Selection Sort. Pero además de eso, también va guardando todos los pasos importantes que se hacen durante el ordenamiento, para luego poder mostrar cómo se comporta el algoritmo paso a paso en una visualización gráfica.

```
    numeros = arreglo.copy()
    pasos = []
    pasos.append((numeros.copy(), -1, -1, None, "Estado inicial"))
```

Aquí lo que se hace es copiar el arreglo que se recibió como entrada, para no modificar el original. Despues, se crea una lista vacía que se llamará `pasos`, donde se irán guardando los distintos estados del arreglo conforme se va ordenando. Justo al inicio, antes de hacer cualquier cosa, se guarda el primer "estado inicial" del arreglo. En ese estado todavía no se ha seleccionado ni comparado nada, por eso los índices van en -1 y el mínimo es `None`.

```
n = len(numeros)
for i in range(n):
    min_idx = i
    pasos.append((numeros.copy(), i, -1, numeros[min_idx], f"Seleccionando desde posición {i}"))
```

Luego se empieza con el ciclo principal, que recorre todo el arreglo elemento por elemento. Cada vez que el ciclo avanza, se asume que el elemento que está en la posición actual `i` es el más pequeño, al menos por ahora. Así que se guarda en `min_idx`. Justo después, se registra un nuevo paso indicando que se está empezando a buscar el valor mínimo desde la posición `i`. Todavía no se está comparando con nadie, pero es como si el algoritmo dijera: "okey, desde aquí voy a buscar el más chico".

```
for j in range(i + 1, n):
    pasos.append((numeros.copy(), i, j, numeros[min_idx], f"Comparando índice {j} con mínimo actual en índice {min_idx}"))
```

Después, entra en un segundo bucle que empieza a comparar el número de la posición `i` con todos los que están adelante de él. Uno por uno. Por cada comparación que hace, se guarda también un paso que indica qué elemento se está comparando (`j`) y con cuál se está comparando (`min_idx`). Esto ayuda mucho cuando se quiere mostrar visualmente cómo el algoritmo "piensa".

```

if numeros[j] < numeros[min_idx]:
    min_idx = j
    pasos.append((numeros.copy(), i, j, numeros[min_idx], f"Nuevo mínimo encontrado: {numeros[min_idx]} en índice {j}"))

```

Ahora, si el número que está revisando en la posición  $j$  resulta ser menor que el que tenía como mínimo hasta ahora, entonces se actualiza  $\text{min\_idx}$ . Es decir, "¡ey! encontré uno más pequeño". Y eso también se guarda como un nuevo paso, indicando que se ha encontrado un nuevo mínimo en otra posición.

```

if i != min_idx:
    numeros[i], numeros[min_idx] = numeros[min_idx], numeros[i]
    pasos.append((numeros.copy(), i, min_idx, numeros[i], f"Intercambiando {numeros[min_idx]} con {numeros[i]}"))

```

Una vez que ya se revisaron todos los elementos restantes y se encontró el valor mínimo real, toca hacer el intercambio. Si el índice mínimo ( $\text{min\_idx}$ ) no es el mismo que el índice actual ( $i$ ), se intercambian sus posiciones. De esta manera, el número más pequeño de esa sección del arreglo queda en su lugar correcto. Obviamente, también se guarda un paso más que diga: "aquí se hizo el cambio", mostrando el estado del arreglo ya actualizado.

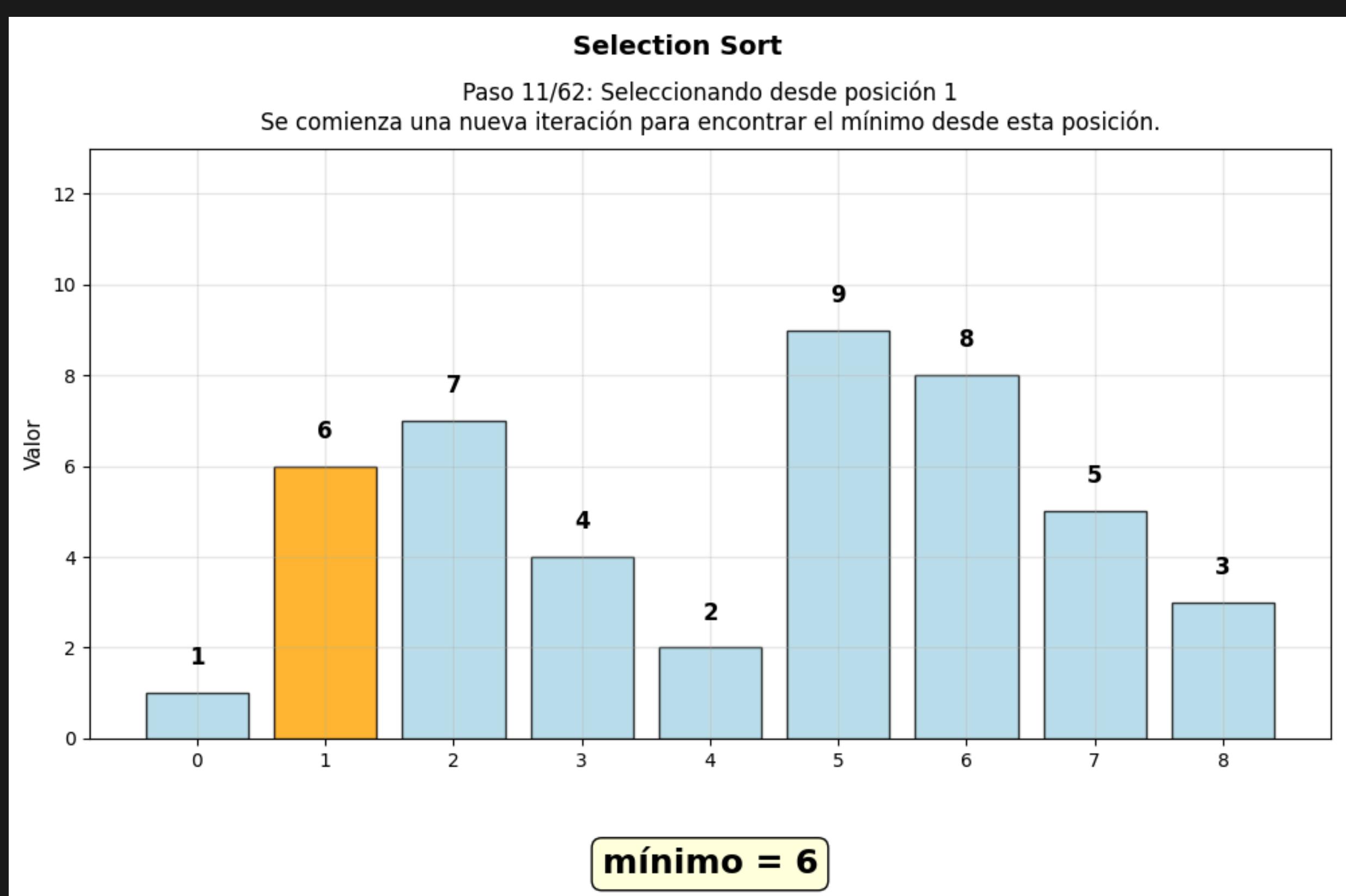
```

pasos.append((numeros.copy(), -1, -1, None, "¡Ordenamiento completado!"))
return pasos

```

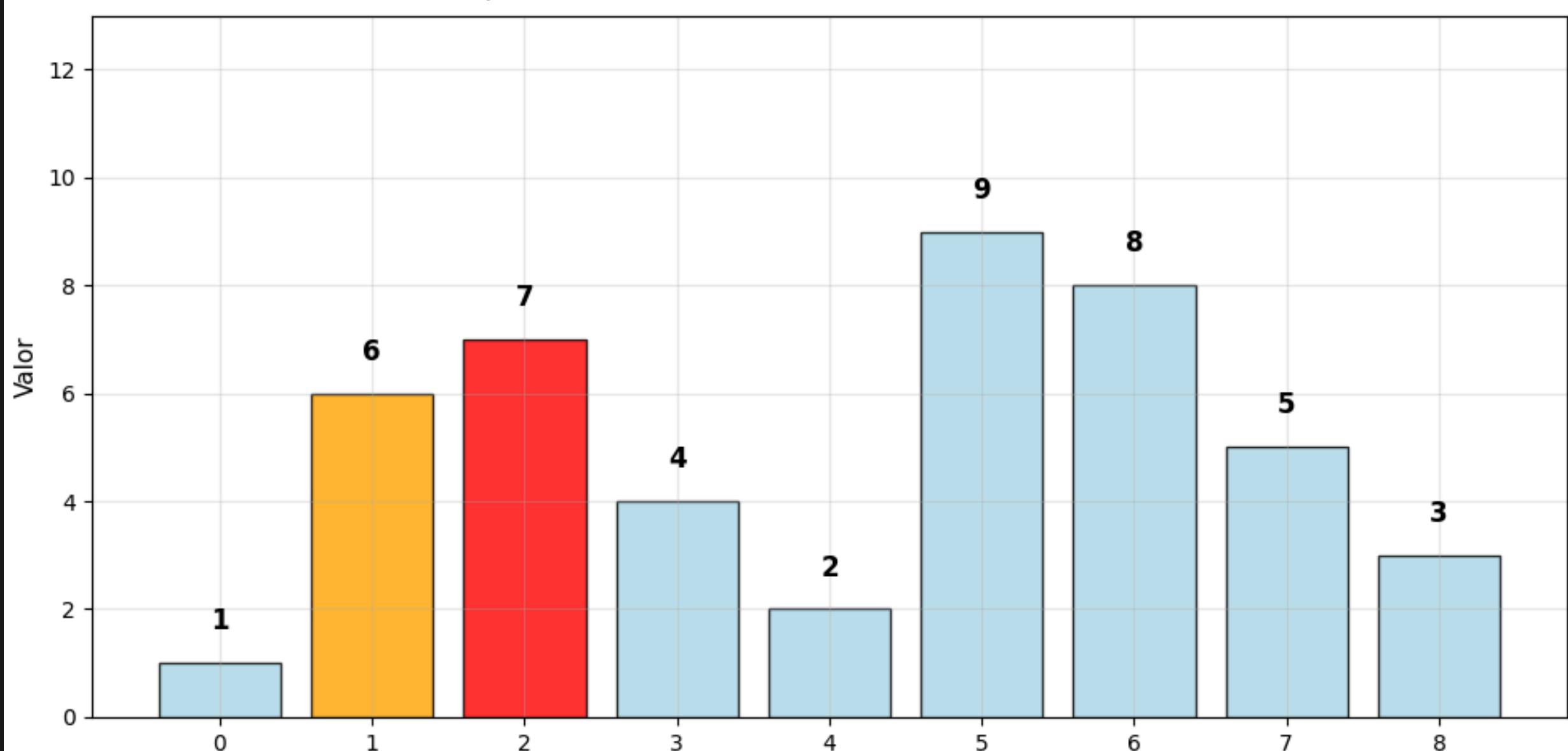
Ya cuando se termina todo el proceso y el arreglo ha quedado ordenado por completo, se agrega un último paso diciendo "¡listo, ya acabamos!". Este paso no tiene ningún índice resaltado ni valor mínimo, simplemente es para marcar el final. Finalmente, se devuelve la lista `pasos`, que contiene todos los momentos clave del proceso y que luego servirán para mostrar el algoritmo en una animación paso a paso.

acá algunas capturas de pantalla sobre la impresión del código:



### Selection Sort

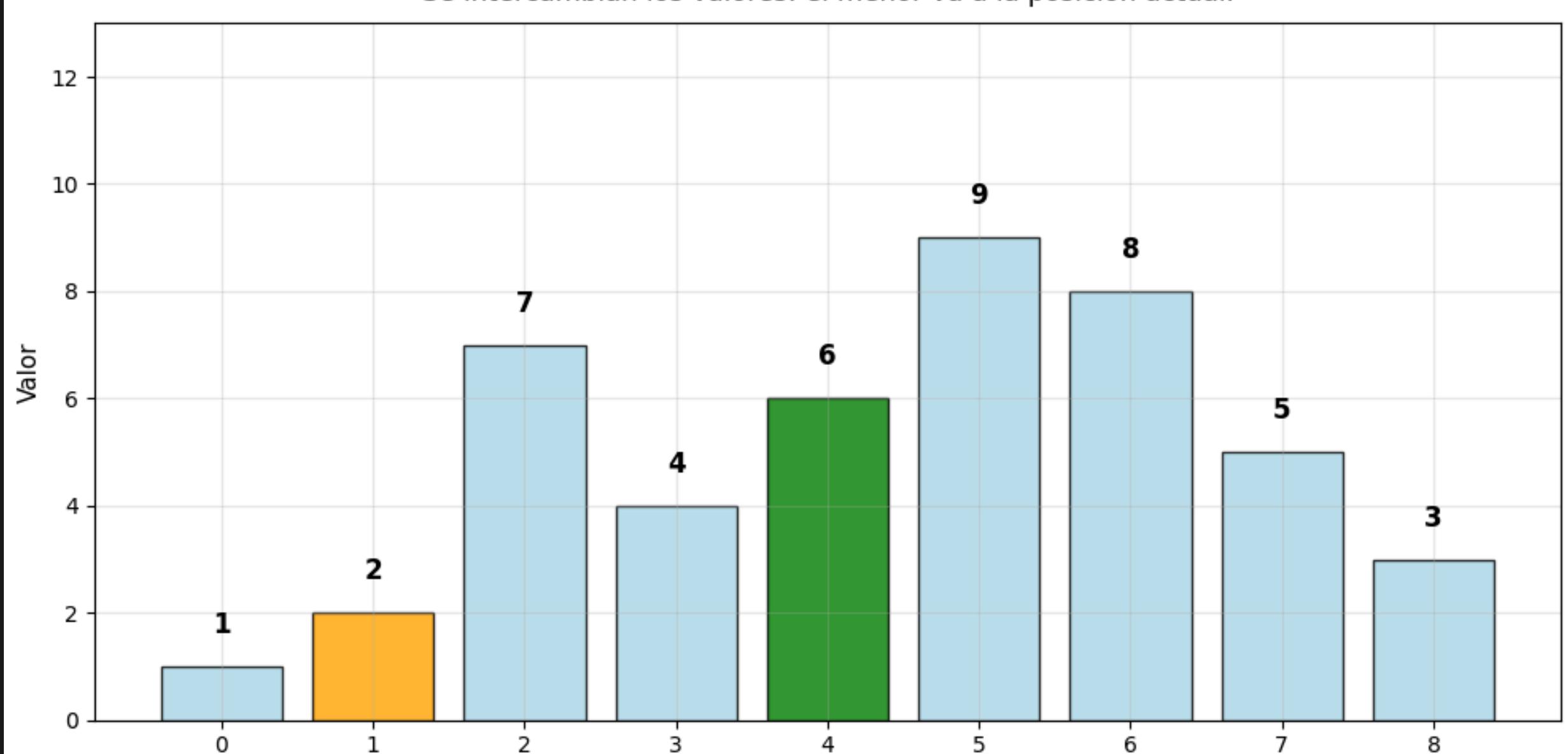
Paso 12/62: Comparando índice 2 con mínimo actual en índice 1  
Se compara el valor actual con el mínimo encontrado hasta ahora.



**mínimo = 6**

### Selection Sort

Paso 21/62: Intercambiando 6 con 2  
Se intercambian los valores: el menor va a la posición actual.



**mínimo = 2**

# En Conclusión:

Después de ver cómo funciona este código de ordenamiento por selección, entendí mucho mejor lo que realmente hace el algoritmo. Ver cada paso animado me ayudó a seguir la lógica, como si el programa me fuera explicando lo que piensa. Me sorprendió cómo algo que parecía tan técnico se puede volver tan visual y fácil de entender. Al final, me di cuenta de que no solo aprendí a ordenar una lista, sino también a valorar lo útil que es mostrar claramente lo que pasa en cada parte del proceso.

