Algoritmo de ordenamiento por burbuja (Bubble Sort) — explicación y ejemplo en Python

El **ordenamiento por burbuja** es un algoritmo sencillo que ordena una lista comparando pares adyacentes y permutándolos cuando están en el orden incorrecto. Repite ese proceso hasta que la lista queda ordenada. Es fácil de entender pero no es eficiente para listas grandes (complejidad promedio $O(n^2)$).

A continuación tienes un ejemplo en Python (versión *optimizada* que corta cuando no hay más intercambios), y luego lo explico **línea por línea**.

```
def bubble_sort(arr):
    """
    Ordena la lista `arr` in-place usando bubble sort optimizado.
    Devuelve la misma lista ordenada para conveniencia.
    """
    n = len(arr)                 # 1
    for i in range(n):           # 2
        swapped = False          # 3
        # Después de i pasadas, los últimos i elementos ya están en su lugar
        for j in range(0, n - i - 1):  # 4
            # Comparar elementos adyacentes
            if arr[j] > arr[j + 1]:    # 5
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  # 6
                swapped = True         # 7
        # Si en una pasada completa no hubo swaps, la lista ya está ordenada
        if not swapped:              # 8
            break                    # 9
    return arr                       # 10

# Ejemplo de uso:
lista = [5, 1, 4, 2, 8]
print("Antes:", lista)
print("Después:", bubble_sort(lista))
```

Explicación línea por línea

Docstring y comentarios iniciales: no son líneas numeradas en el código pero ayudan a entender la función.

1. `n = len(arr)`
   - Calcula la cantidad de elementos de la lista y la guarda en `n`. Lo usamos para saber cuántas comparaciones y pasadas necesitamos.
2. `for i in range(n):`
   - Bucle que hace hasta `n` pasadas sobre la lista. En la práctica, con la optimización (flag `swapped`) puede terminar antes. `i` representa cuántas pasadas completas ya se hicieron.
3. `swapped = False`
   - Inicializa un indicador que nos permite detectar si en la pasada actual se hizo al menos un intercambio. Si no hay intercambios, la lista ya está ordenada y podemos terminar temprano.
4. `for j in range(0, n - i - 1):`
   - Bucle interno que recorre los índices desde `0` hasta `n - i - 2`.
   - Razón: después de `i` pasadas los `i` últimos elementos ya están en su posición correcta (los más grandes "burbujearon" hacia el final), así que no hace falta compararlos otra vez.

5. `if arr[j] > arr[j + 1]:`
    - Compara el elemento actual `arr[j]` con el siguiente `arr[j+1]`. Si el actual es mayor que el siguiente (orden ascendente), hay que intercambiarlos.
6. `arr[j], arr[j + 1] = arr[j + 1], arr[j]`
    - Intercambia los dos elementos adyacentes en una sola línea (tupla desempaquetada). Ahora el menor de los dos queda en la posición `j`.
7. `swapped = True`
    - Marca que ocurrió al menos un intercambio en esta pasada. Eso significa que quizás aún no está totalmente ordenado.
8. `if not swapped:`
    - Comprueba si `swapped` sigue siendo `False` después de revisar todos los pares en la pasada.
9. `break`
    - Si no hubo intercambios durante la pasada completa, la lista ya está ordenada y salimos del bucle externo para evitar trabajo inútil.
10. `return arr`
    - Devuelve la lista ya ordenada (la ordena in-place, es decir modifica la lista original, pero devolverla es conveniente para imprimir o encadenar).

Ejemplo paso a paso (lista = [5, 1, 4, 2, 8])

- Pasada 1 (i=0): comparar y mover máximos hacia la derecha
    - [5,1] → swap → [1,5,4,2,8]
    - [5,4] → swap → [1,4,5,2,8]
    - [5,2] → swap → [1,4,2,5,8]
    - [5,8] → no swap → [1,4,2,5,8]
      Resultado final de la pasada 1: `[1,4,2,5,8]`
- Pasada 2 (i=1):
    - [1,4] → no swap
    - [4,2] → swap → [1,2,4,5,8]
    - [4,5] → no swap
      Resultado: `[1,2,4,5,8]`
- Pasada 3 (i=2):
    - [1,2] → no swap
    - [2,4] → no swap
      Como no hubo swaps, el algoritmo detecta `swapped = False` y rompe: la lista ya está ordenada.