Bubble Sort

Varun Kumar

July 4, 2025

1. Logic

Bubble Sort is a comparison-based sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. The largest unsorted element ”bubbles” to the correct position in each pass.

key Idea

After the i^{th} pass, the i^{th} largest elements is at it’s sorted position.

2. Number of Comparisons and Swaps

Let n be the number of elements.

Worst Case (Reverse Sorted)

- Comparisons: $\frac{n(n-1)}{2}$
- Swaps: $\frac{n(n-1)}{2}$

Best Case (Already Sorted)

- Comparisons: $n-1$ (with optimization)
- Swaps: 0

3. Optimal Bubble Sort (Early Termination)

Introduce a flag to detect if any swaps happened during the pass. If no swaps occur, the array is already sorted.

4. Pseudocode

```
function bubbleSort(arr):
    n = length(arr)
    for i = 0 to n-1:
        swapped = false
        for j = 0 to n-i-2:
            if arr[j] > arr[j+1]:
                swap arr[j] and arr[j+1]
                swapped = true
        if not swapped:
            break
```

Stability Note

Bubble Sort is stable by default because it swaps elements only when `arr[j] > arr[j+1]`.
If we change the condition to `arr[j]` $\geq$ `arr[j+1]`, it may swap equal elements, making the sort unstable.

5. Example Walkthrough

Given: `[5, 1, 4, 2, 8]`

Pass 1

`[5, 1, 4, 2, 8]` $\Rightarrow$ `[1, 5, 4, 2, 8]` $\Rightarrow$ `[1, 4, 5, 2, 8]` $\Rightarrow$ `[1, 4, 2, 5, 8]`

Pass 2

`[1, 4, 2, 5, 8]` $\Rightarrow$ `[1, 2, 4, 5, 8]`

Pass 3

`[1, 2, 4, 5, 8]` → No swaps → Done

6. Python Code with Explanation

```
def bubble_sort(arr):
    # Get the length of the list
    n = len(arr)

    # Traverse the list n times
    for i in range(n):
        # Initialize swapped flag as False at the beginning of each pass
        swapped = False

        # Perform comparisons up to the unsorted portion (n - i - 1)
        for j in range(0, n - i - 1):
            # Compare adjacent elements
            if arr[j] > arr[j + 1]:
                # Swap if they are in the wrong order
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                # Set flag to True to indicate a swap happened
                swapped = True

        # If no elements were swapped during the inner loop, the list is
            sorted
        if not swapped:
            break

    # Return the sorted list
    return arr
```

7. Time & Space Complexity and It's Properties

Case	Complexity	Property	Value
Best Case	$O(n)$	Stable	Yes
Average Case	$O(n^2)$	In-place	Yes
Worst Case	$O(n^2)$	Adaptive	Yes (optimized)
Space Complexity	$O(1)$ (in-place)	Recursive	No