BUBBLE SORT

1. Definition

Bubble Sort is a simple comparison-based sorting algorithm that repeatedly compares adjacent elements and swaps them if they are out of order. After each full pass, the largest (or smallest) element "bubbles" to its correct position.

2. How It Works (Step-by-Step Idea)

1. Compare **arr[i]** and **arr[i+1]**.
2. If they are in the wrong order, **swap them**.
3. Continue until the end of the array → one full pass done.
4. The largest element moves to the last position.
5. Repeat for the remaining unsorted part.
6. Stop early if no swaps occurred during a pass (optimization).

3. Example

Input: `[5, 3, 8, 4, 2]`

Pass 1:

- 5 & 3 → swap → `[3, 5, 8, 4, 2]`
- 5 & 8 → OK
- 8 & 4 → swap → `[3, 5, 4, 8, 2]`
- 8 & 2 → swap → `[3, 5, 4, 2, 8]`

The largest element *8* settles at last.

Pass 2:

- `[3, 5, 4, 2, 8]` → continue comparisons → ends with `[3, 4, 2, 5, 8]`

Pass 3:

- `[3, 4, 2, 5, 8]` → ends with `[3, 2, 4, 5, 8]`

Pass 4:

- `[3, 2, 4, 5, 8]` → ends with `[2, 3, 4, 5, 8]`

Sorted!

4. Algorithm (Simple Version)

```
BubbleSort(arr, n)
    for i = 0 to n-1
        for j = 0 to n-i-2
            if arr[j] > arr[j+1]
                swap arr[j], arr[j+1]
```

5. Algorithm (Optimized Version – Uses Flag)

```
BubbleSort(arr, n)
    for i = 0 to n-1
        swapped = false
        for j = 0 to n-i-2
            if arr[j] > arr[j+1]
                swap arr[j], arr[j+1]
                swapped = true
        if swapped == false
            break   // array already sorted
```

6. Time Complexity

Case	Complexity
Best	**O(n)** – when array already sorted (optimized version)
Average	$O(n^2)$
Worst	$O(n^2)$

8. Stability

✔ **Stable Sorting Algorithm**
Because equal elements stay in their original order.

9. Adaptability

✔ Adaptive when optimized (stops early if sorted).
✘ Non-adaptive in basic version.

10. When to Use Bubble Sort

- Teaching and learning sorting basics
- Very small datasets
- When simplicity is more important than speed
- When array is *almost sorted* (optimized version performs well)

11. When NOT to Use Bubble Sort

- Large datasets
- Performance-critical systems
- Real-time applications

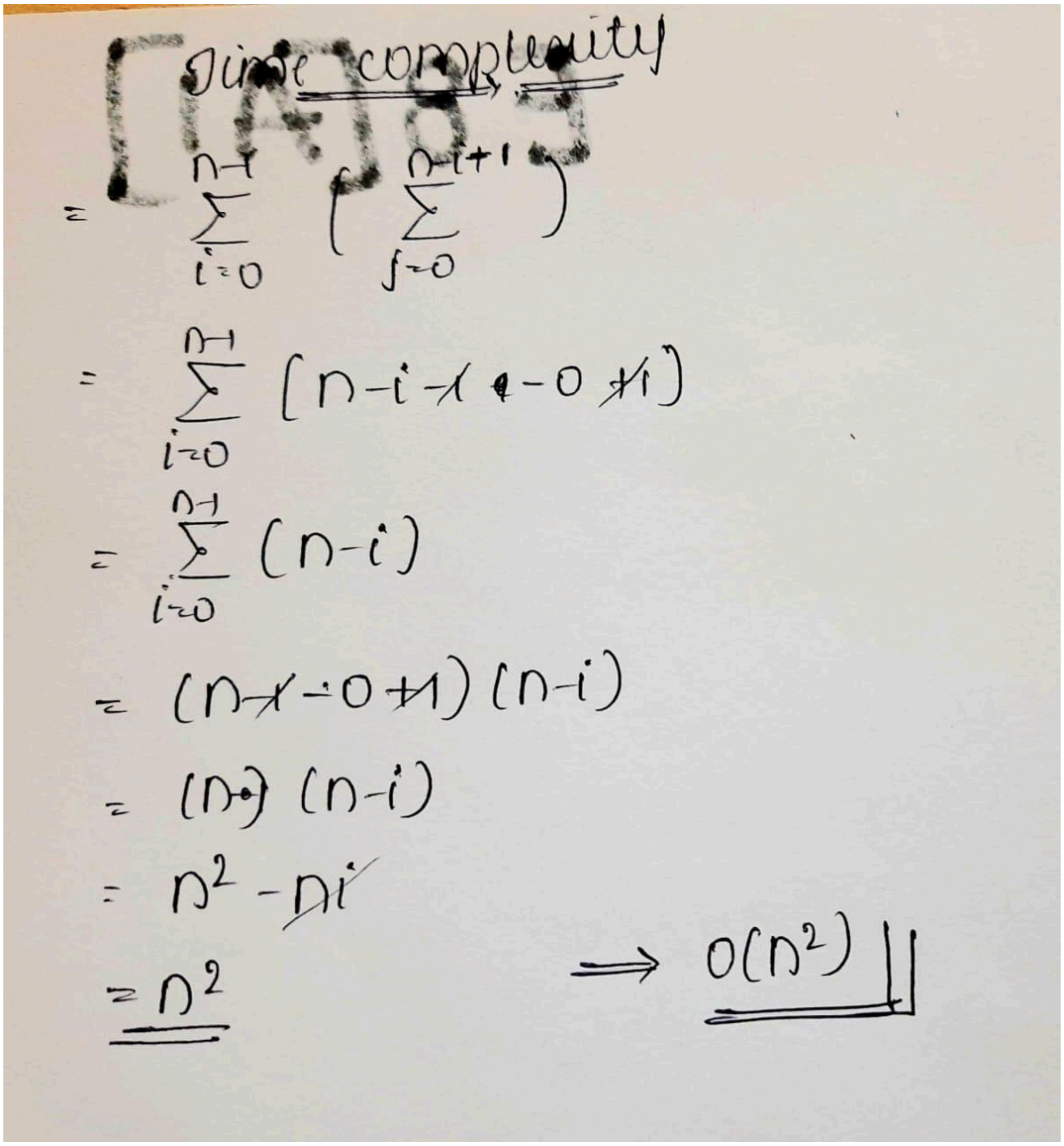
12. Advantages

- Easy to understand and implement
- Stable
- No extra memory needed
- Works well on nearly sorted data (optimized)

13. Disadvantages

- Very slow for large datasets
- Performs unnecessary comparisons without optimization
- Not suitable for competitive programming or real applications

Efficiency Analysis



Time complexity

$$= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-i+1} \right)$$

$$= \sum_{i=0}^{n-1} (n-i-\cancel{1}-0\cancel{+1})$$

$$= \sum_{i=0}^{n-1} (n-i)$$

$$= (n-\cancel{1}-0\cancel{+1})(n-i)$$

$$= (n)(n-i)$$

$$= n^2 - \cancel{ni}$$

$$= \underline{\underline{n^2}} \qquad \Longrightarrow \qquad \underline{\underline{O(n^2)}}$$

Component	Worst Case Value
Total Passes	**$n - 1$**
Total Comparisons	**$n(n - 1) / 2$**
Total Swaps	**$n(n - 1) / 2$**
Time Complexity	**$O(n^2)$**