

2

Bubble Sort

EASY

What You Will Learn

- ✓ Why it's called 'Bubble' Sort
- ✓ How adjacent swaps push the largest element to the end each pass
- ✓ The early-exit optimization that makes it $O(n)$ in best case
- ✓ Why it's rarely used in practice

Intuition First

Imagine bubbles in water. Heavy things sink, light things float up. In each pass, the largest unsorted element 'bubbles up' to its correct position at the right end. After n passes, you're done.

The Core Idea

- Compare adjacent pairs ($arr[j]$ and $arr[j+1]$)
- If $left > right$, swap them
- After each pass, the biggest remaining element is in its final spot
- Repeat for $n-1$ passes (each pass has one less comparison)

Visual Explanation

Pass 1: 5 bubbles to the right end

Compare 5 & 3 → swap

5	3	8	4	2
---	---	---	---	---

Compare 5 & 8 → no swap

3	5	8	4	2
---	---	---	---	---

Compare 8 & 4 → swap

3	5	8	4	2
---	---	---	---	---

Compare 8 & 2 → swap

3	5	4	8	2
---	---	---	---	---

Pass 1 done — 8 is in place ✓

3	5	4	2	8
---	---	---	---	---

■ ■ How It Works — Step by Step

1. Outer loop: i goes from 0 to $n-2$
2. Inner loop: j goes from 0 to $n-2-i$ (last i elements already sorted)
3. If $arr[j] > arr[j+1]$: swap them
4. After each inner loop, one more element is in its final position
5. Optimization: if no swap happened in a full pass, array is already sorted — stop early

■ Dry Run on [5, 3, 8, 4, 2]

Step 1 Pass 1 complete — 8 bubbled to end

3	5	4	2	8
---	---	---	---	---

Step 2 Pass 2 complete — 5 bubbled to pos 3

3	4	2	5	8
---	---	---	---	---

Step 3 Pass 3 complete — 4 bubbled to pos 2

3	2	4	5	8
---	---	---	---	---

Step 4 Pass 4 complete — done!

2	3	4	5	8
---	---	---	---	---

■ Java Implementation (with optimization)

```
public static void bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        boolean swapped = false; // Early exit flag
        for (int j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap adjacent elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}
```

```
}  
  
}  
  
// If no swap in this pass, array is sorted  
if (!swapped) break;  
  
}  
  
}
```

■ ■ Time & Space Complexity

Case	Time	Space	What happens?
Best	$O(n)$	$O(1)$	Already sorted — early exit fires after 1 pass
Average	$O(n^2)$	$O(1)$	Random order — many swaps needed
Worst	$O(n^2)$	$O(1)$	Reverse sorted — maximum comparisons & swaps

■ ■ Common Mistakes

- Inner loop should go to $n-1-i$, not $n-1$ (you'd recheck sorted elements)
- Forgetting the swapped flag — without it, no best case improvement
- Bubble sort is STABLE — equal elements keep their order (unlike Selection)

■ Pro Tips

- The swapped flag is the key optimization — always include it
- Bubble sort is great for teaching, but rarely used in production
- It's stable — one advantage over selection sort

■ Practice Problems

- [Easy] Trace bubble sort on [9, 4, 2, 8, 1] — count total swaps
- [Easy] What input gives best case $O(n)$?
- [Med] Sort strings alphabetically using bubble sort logic
- [Med] Write a version that sorts from right to left

■ Quick Revision

- Compare adjacent pairs, swap if out of order

- Largest element bubbles to end each pass
- Best: $O(n)$ with swapped flag | Worst: $O(n^2)$
- $O(1)$ space — stable sort
- Rarely used in practice due to $O(n^2)$ average