

3

Insertion Sort

EASY

What You Will Learn

- ✓ How insertion sort mimics sorting playing cards in your hand
- ✓ Why it's excellent for nearly-sorted arrays
- ✓ How it achieves $O(n)$ in best case
- ✓ The 'shifting' mechanism instead of swapping

Intuition First

Think about sorting playing cards in your hand. You pick one card, then slide it left into the right spot among the cards you've already sorted. That's exactly what insertion sort does — one element at a time, inserted into its correct position.

The Core Idea

- Divide array into sorted (left) and unsorted (right) sections
- Pick first element of unsorted section — call it 'key'
- Shift sorted elements rightward until you find key's correct spot
- Insert key there. Move boundary right by 1.

Visual Explanation — Cards Analogy

Sorted hand → [3] New card → 1
Shift 3 right → [] [3] Insert 1 → [1] [3] ✓

Sorted: [3] | Key: 1

3	1	4	1	5
---	---	---	---	---

→ 3 > 1, shift 3 right, insert 1 before 3

Sorted: [1,3] | Key: 4

1	3	4	1	5
---	---	---	---	---

→ 3 < 4, no shift, insert 4 here

Sorted: [1,3,4] | Key: 1

1	3	4	1	5
---	---	---	---	---

Final: sorted ✓



■ Java Implementation

```
public static void insertionSort(int[] arr) {
    int n = arr.length;
    for (int i = 1; i < n; i++) {
        int key = arr[i]; // Element to be inserted
        int j = i - 1;

        // Shift elements right until correct position found
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j]; // Shift right (not a swap!)
            j--;
        }
        arr[j + 1] = key; // Place key in its correct spot
    }
}
```

■ Time & Space Complexity

Case	Time	Space	What happens?
Best	$O(n)$	$O(1)$	Already sorted — inner while never executes
Average	$O(n^2)$	$O(1)$	Each element shifts $\sim n/2$ positions on average
Worst	$O(n^2)$	$O(1)$	Reverse sorted — every element shifts fully

■ Key Insight — Shifts vs Swaps

- Insertion sort SHIFTS, not swaps (one assignment vs three)
- This makes it faster in practice than bubble/selection for small arrays
- It's the preferred sort for small n or nearly-sorted data

■ Common Mistakes

- Loop starts at $i=1$, not $i=0$ (index 0 is already 'sorted')
- Don't use swap — use shift: `arr[j+1] = arr[j]`
- Remember: `arr[j+1] = key` at the end, not `arr[j]`

■ Practice Problems

[Easy] Trace insertion sort on [5, 2, 4, 6, 1, 3]

[Easy] How many shifts needed for [3, 2, 1]?

[Med] Binary Insertion Sort — use binary search to find insertion point

[Med] Show insertion sort is stable using an example with duplicates

■ Quick Revision

- Pick key from unsorted section, shift sorted elements right
- Best: $O(n)$ for sorted arrays — loop's while barely runs
- Worst: $O(n^2)$ for reverse sorted
- Stable, in-place, adaptive
- Best algorithm for small arrays or nearly-sorted input