Insertion Sort

Insertion Sort is a **simple comparison-based sorting algorithm**.
It works the same way as how we sort **playing cards in our hands**:

- Start with one card (already sorted hand).
- Pick the next card and insert it into the correct position in the sorted hand.
- Repeat until all cards are placed in sorted order.

🔹 How Insertion Sort Works (Step by Step)

1. Assume the **first element** is already sorted.
2. Take the next element (key) and compare it with the sorted part.
3. Shift all elements greater than the key to the right.
4. Insert the key into the correct position.
5. Repeat until the entire array is sorted.

🔹 Example (Ascending Order)

Sort the array **[5, 3, 4, 1, 2]** using Insertion Sort:

- **Pass 1:** Key = 3 → Compare with 5 → Insert before 5 → `[3, 5, 4, 1, 2]`
- **Pass 2:** Key = 4 → Compare with 5 → Insert before 5 → `[3, 4, 5, 1, 2]`
- **Pass 3:** Key = 1 → Compare with 5, 4, 3 → Insert at beginning → `[1, 3, 4, 5, 2]`

- **Pass 4:** Key = 2 → Compare with 5, 4, 3 → Insert after 1 → `[1, 2, 3, 4, 5]`

Sorted array = `[1, 2, 3, 4, 5]`

🔹 Algorithm (Ascending Order)

```
for i = 1 to n-1
    key = arr[i]
    j = i - 1
    while j >= 0 and arr[j] > key
        arr[j+1] = arr[j]
        j = j - 1
    arr[j+1] = key
```

🔹 Time Complexity

- **Best Case:** O(n) → when array is already sorted
- **Average Case:** O(n²)
- **Worst Case:** O(n²) → when array is sorted in reverse order

🔹 Space Complexity

- **O(1)** (In-place sorting, no extra array needed)

🔹 Characteristics

Easy to implement
Stable (keeps relative order of equal elements)
Efficient for **small arrays** or **nearly sorted arrays**
Inefficient for large datasets compared to advanced algorithms

Real-life Analogy:

Think of sorting **playing cards**: You take one card at a time and insert it into its proper place among the already sorted cards in your hand.