Insertion Sort

What is Insertion Sort?

Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time by taking an element and inserting it into its correct position.

- **Time Complexity:**
    - **Best case (Already Sorted):** O(n)
    - **Average case:** O(n^2)
    - **Worst case (Reverse Sorted):** O(n^2)
- **Space Complexity:** O(1)O(1) (In-place sorting)
- **Stable Sorting Algorithm:** Yes (Preserves the order of duplicate elements)

How Insertion Sort Works?

1. Start from the **second element** (index 1) and compare it with previous elements.
2. If the current element is **smaller**, shift the larger elements **one position to the right**.
3. Insert the current element in the correct position.
4. Repeat until the array is sorted.

Dry Run Example

Input: [12, 11, 13, 5, 6]

Sorting Process:

Pass	Current Element	Array After Sorting
1	11	**[11, 12, 13, 5, 6]**
2	13	**[11, 12, 13, 5, 6]** (No change)
3	5	**[5, 11, 12, 13, 6]**
4	6	**[5, 6, 11, 12, 13]** (Final sorted)

Q1: What is the best-case time complexity of Insertion Sort?

Answer: O(n) (When the array is already sorted, only one comparison per element is needed.)

Q2: What is the worst-case time complexity of Insertion Sort?

Answer: O(n^2) (When the array is sorted in reverse order, each element has to be compared and shifted.)

Q3: Why is Insertion Sort better for small datasets?

Answer: Since it performs well for **nearly sorted data** and **small datasets**, it is used when efficiency is not a major concern.

Q4: Is Insertion Sort a stable sorting algorithm?

Answer: Yes, because it **does not swap equal elements**, maintaining their relative order.

Q5: Can we optimize Insertion Sort?

Answer:

- **Binary Insertion Sort:** Instead of linear search, use **Binary Search** to find the correct position. This reduces comparisons but **not swaps**.
- **Shell Sort:** A variation of Insertion Sort that sorts elements at a **gap distance**, improving efficiency.

When to Use Insertion Sort?

Best Suited For:

- Small datasets (as it has minimal overhead).
- Nearly sorted data (fastest in such cases).
- Sorting data as it arrives (real-time processing).

Not Suitable For:

- Large datasets (Quadratic time complexity makes it slow).

Conclusion

- **Insertion Sort is simple and efficient for small or nearly sorted datasets.**
- **It is stable and in-place.**
- **For larger datasets, Merge Sort or Quick Sort is better.**