LLD Practice Problem:

Sorting Context

Problem: (Easy)

Functional Requirements

1. **Pluggable Sorting Algorithms**
   - Make a class with the name **SortHandler** having method called **sort(...)** that can sort an input integer array.
   - The **sort(...)** method of **SortHandler** should not sort the input array by itself but should delegate this request to a **SortingStrategy** class.
   - **SortingStrategy** should have multiple concrete sorting strategies like **QuickSortStrategy** and **MergeSortStrategy**, each independently selectable at runtime. The **sort(...)** method of **SortHandler** should just delegate this request to one of the strategies chosen.
   - Keep a reference of the Sorting strategy chosen in a variable with name **sortingStrategy** in **SortHandler** class that initializes any sorting strategy to **sortingStrategy** during Object creation (using constructor).
   - Quick-Sort, should support two variants:
     • Normal Quick-Sort (pivot = last element)
     • Randomized Quick-Sort (pivot = random element)
   - Merge-Sort, should support two variants:
     • Normal Merge-Sort (uses auxiliary arrays)
     • In-Place Merge-Sort (merges within the original array).
2. **Order Direction**
   - Overload the `sort(...)` method for each strategy, one taking just an input array and another taking an extra string parameter to define the order of sorting (ascending / descending).

Non-Functional Requirements

1. **Extensibility**
   - It must be easy to add new algorithms (e.g. Heap-Sort) without modifying existing code (Open/Closed).
2. **Plug and Play Model**
   - The strategies should be plug and playable easily.
3. **Performance**
   - Sorting large datasets should remain efficient (average O(n log n) time).

Expectations:
UML + Working Code.