1. Definition: What is the Merge Intervals Pattern?

The **Merge Intervals** pattern is a common algorithmic approach used when dealing with a collection of intervals (usually represented as pairs like [start, end]) and you need to:

- Merge all overlapping intervals into one,
- Or process the intervals in a way that their overlaps are efficiently handled.

A classic example:

Given intervals [[1,3],[2,6],[8,10],[15,18]], after merging overlaps, the result is [[1,6],[8,10],[15,18]].

2. Where Can It Be Applied?

The pattern is applied in problems that involve **ranges**, **segments**, or **intervals** where overlaps need to be managed.

Common scenarios include:

- Calendar event merging (combine overlapping meetings).
- Finding free time slots from booked schedules.
- Merging busy intervals in CPU scheduling.
- Range compression (e.g., summarize address or time ranges).
- Data compression, where data is stored as ranges instead of individual elements.
- Merging intervals in genome sequencing or network IP ranges.

3. How to Apply the Merge Intervals Pattern

Here's the typical approach:

Step 1: Sort all intervals by their start time.

Why? Overlaps can only happen between intervals that are next to each other after sorting.

Step 2: Iterate through intervals and merge when overlapping.

- Initialize a result list with the first interval.
- For each next interval, compare it to the last one in the result:
  - **If current interval's start ≤ last interval's end:** Overlap exists; merge them by updating the end.
  - **Else:** No overlap; add the current interval to the result.

Pseudocode:

```
List<int[]> merge(List<int[]> intervals) {
   sort intervals by start;
   result = new empty list;
   for (interval in intervals) {
      if (result is empty or result.last.end < interval.start)
         add interval to result;
      else
         result.last.end = max(result.last.end, interval.end);
   }
```

```
    return result;
}
```

4. How to Recognize Questions That Need This Pattern

Look for problems where:

- You're given a list/array of intervals or ranges.
- The task involves merging, covering, minimizing overlaps, or finding gaps.
- Words like "merge," "combine," "overlap," "conflict," "meeting," "event," or "calendar" are in the problem.

Examples of such problems:

- Merge overlapping intervals.
- Insert a new interval and merge if necessary.
- Find intersections of intervals.
- Minimum number of meeting rooms required.
- Employee free time.
- Remove covered intervals.

5. Benefits of the Merge Intervals Pattern

- **Efficiency**: Sorting + linear scan = O(N log N), much faster than brute force pairwise comparison.
- **Simplicity**: Once intervals are sorted, you never look back—just process in order.
- **Versatility**: Can be adapted for related problems (finding overlaps, gaps, union, intersection, etc.).

Summary Table

Step	Action
1. Sort	Sort intervals by start (and optionally end if needed)
2. Iterate	Compare current interval with the last merged interval
3. Merge	Merge if overlapping, otherwise add as a new interval
4. Return	Return the result list of merged intervals

#	Problem Name	Platform	Link

1	Merge Intervals	LeetCode	LeetCode 56
2	Insert Interval	LeetCode	LeetCode 57
3	Meeting Rooms II	LeetCode	LeetCode 253
4	Meeting Rooms	LeetCode	LeetCode 252
5	Interval List Intersections	LeetCode	LeetCode 986
6	Employee Free Time	LeetCode	LeetCode 759
7	Minimum Number of Arrows to Burst Balloons	LeetCode	LeetCode 452
8	Non-overlapping Intervals	LeetCode	LeetCode 435
9	Remove Covered Intervals	LeetCode	LeetCode 1288
10	Find Right Interval	LeetCode	LeetCode 436
11	Range Module	LeetCode	LeetCode 715
12	Summary Ranges	LeetCode	LeetCode 228
13	Car Pooling	LeetCode	LeetCode 1094
14	Interval Intersection	LeetCode	LeetCode 986 (Same as #5)
15	Merge Sorted Interval Lists	LeetCode Discuss	Merge K Sorted Interval Lists

16	**CPU Scheduling (Minimum Number of CPUs to Finish Tasks)**	LeetCode	Task Scheduler