Introduction to Merge Intervals

Let’s go over the Merge Intervals pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

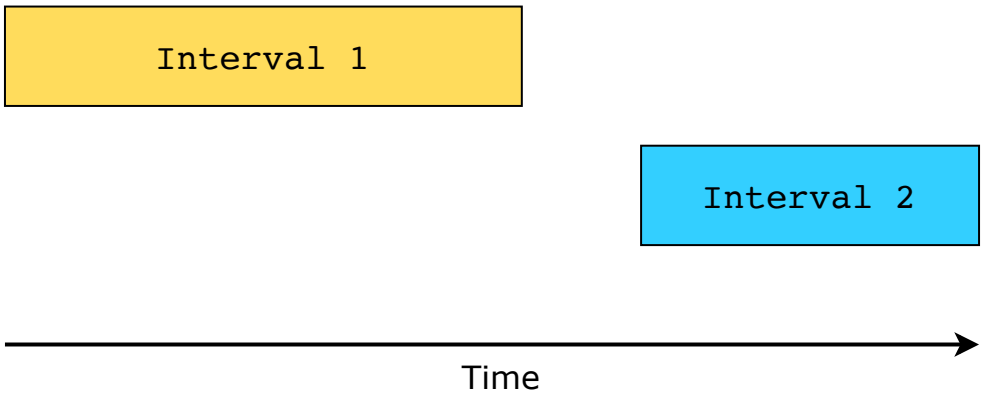
About the pattern

The **merge intervals** pattern deals with problems involving overlapping intervals. Each interval is represented by a start and an end time. For example, an interval of $[10, 20]$ seconds means that the interval starts at 10 seconds and ends at 20 seconds. This pattern involves tasks such as merging intersecting intervals, inserting new intervals into existing sets, or determining the minimum number of intervals needed to cover a given range. The most common problems solved using this pattern are event scheduling, resource allocation, and time slot consolidation.

The key to understanding this pattern and exploiting its power lies in understanding how any two intervals may overlap. The illustration below shows different ways in which two intervals can relate to each other:

1. Nonoverlapping intervals

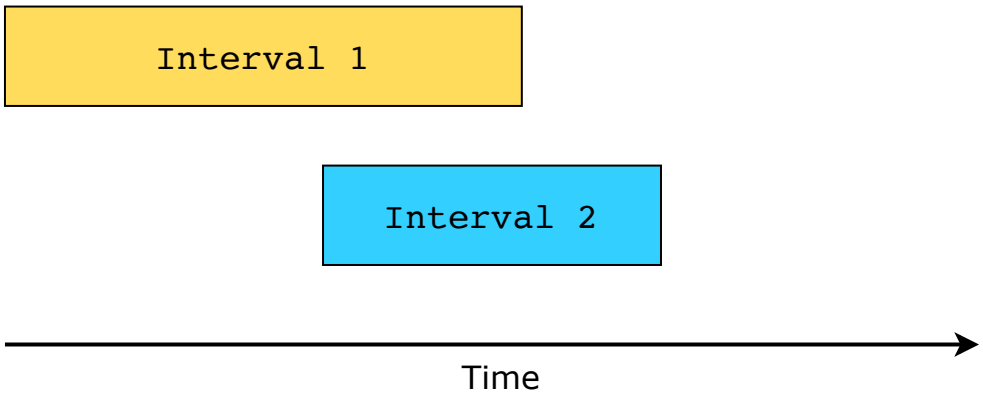
The second interval starts after the end of the first interval.



1 of 3

2. Partially overlapping intervals

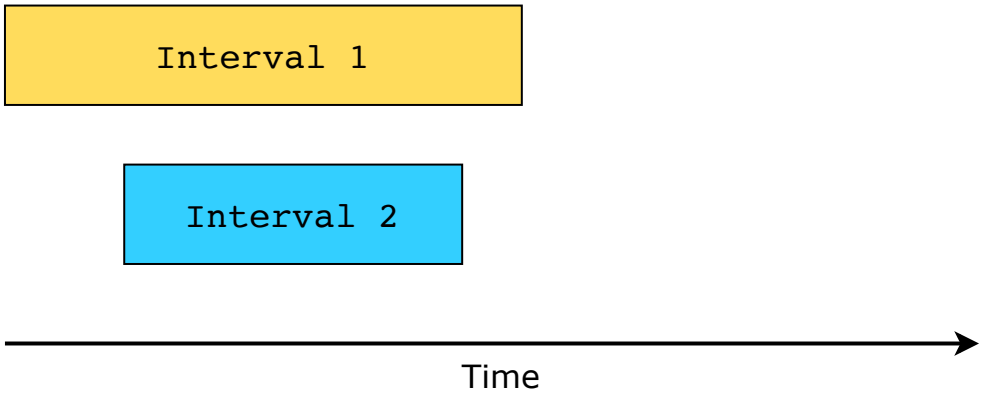
The second interval starts before the end of the first interval, and ends after the end of the first interval.



2 of 3

3. Completely overlapping intervals

The second interval starts after the first interval has started, and ends before the end of the first interval.



3 of 3

Examples

The following examples illustrate some problems that can be solved with this approach:

1. **Merge intervals:** Given a sorted list of intervals, merge all overlapping intervals.

Given the following sorted list of intervals, we need to merge all overlapping intervals into an output list.

Input	[1, 4]	[3, 7]	[9, 12]	[10, 14]	[15, 17]	[16, 18]

1 of 8

2. **Meeting rooms**: Given an array of meeting time intervals consisting of start and end times, determine if a person could attend all meetings.

For the following list of meeting times, we need to check whether a person could attend all meetings.

The idea here is to sort the meetings with respect to the starting time. Then, iterate over the meetings and check whether each meetings ends before the start of the next meeting.

Input	[9, 10]	[15, 16]	[5, 7]	[1, 4]	[12, 14]

1 of 7

Does your problem match this pattern?

Yes, if both of these conditions are fulfilled:

- **Array of intervals:** The input data is an array of intervals.
- **Overlapping intervals:** The problem requires dealing with overlapping intervals, either to find their union, their intersection, or the gaps between them.

Real-world problems

Many problems in the real world use the merge intervals pattern. Let's look at some examples.

- **Display busy schedule:** Display the busy hours of a user to other users without revealing the individual meeting slots in a calendar.
- **Schedule a new meeting:** Add a new meeting to the tentative meeting schedule of a user in such a way that no two meetings overlap each other.
- **Task scheduling in operating systems (OS):** Schedule tasks for the OS based on task priority and the free slots in the machine's processing schedule.

Strategy time!#

Match the problems that can be solved using the merge intervals pattern.

> **Note**: Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side

Find the 3^{rd} closest point to the origin.	Merge Intervals
Schedule three interviews for an interviewer in one	Some other pattern

day.

For a football tournament, find the durations during which more than one games are being played simultaneously.

Given an unsorted array of integers, find the length of the longest increasing subsequence.

Reset Show Solution Submit