**Intervals**

This page covers problems involving intervals, which are given as a list of [start, end] times.A screenshot of a computer

AI-generated content may be incorrect.

Interval problems typically involve sorting the given intervals, and then processing each interval in sorted each order. On this page, we'll cover:

1. Sorting intervals by start time
2. Sorting intervals by end time

**Sorting by Start Time**

Sorting intervals by their start times **makes it easy to merge two intervals that are overlapping.**

A screenshot of a computer

AI-generated content may be incorrect.

Merging two overlapping intervals.

**Overlapping Intervals**

After sorting by start time, an interval overlaps with the previous interval if it starts before the end time of the previous interval.A screenshot of a computer

AI-generated content may be incorrect.

Detecting overlapping intervals is the basis of the question [**Can Attend Meetings**](https://www.hellointerview.com/learn/code/intervals/can-attend-meetings), in which we are given a list of intervals representing the start and end times of meetings, and we need to determine if a person can attend all meetings.

We sort the intervals by their start times and iterate over each meeting. If the current meeting overlaps with the previous one, we return False. If we make it through the entire list without finding any overlaps, we return True.

**Merging Intervals**

When an interval overlaps with the previous interval in a list of intervals sorted by start times, they can be merged into a single interval.

To merge an interval into a previous interval, we set the end time of the previous interval to be the max of either end time.

prev\_interval[1] = max(prev\_interval[1], interval[1])

In [**Merge Intervals**](https://www.hellointerview.com/learn/code/intervals/merge-intervals), we are given a list of intervals and need to return a list with all overlapping intervals merged together. We create a new list containing the merged intervals, sort the intervals by their start times, and then iterate over each interval. If the current interval overlaps with the last interval in the merged list, we merge the current interval into the last interval in the merged list. Otherwise, we add the current interval to the merged list.

Top of Form

**Sorting by End Time**

To see why we sometimes want to sort by end times instead of start time, let's consider the question of finding the maximum number of non-overlapping intervals in a given list of intervals.

Our solution will sort the intervals, and then greedily try to add each interval to the set of non-overlapping intervals.

If we sort by start time, we risk adding an interval that starts early but ends late, which will block us from adding other intervals until that interval ends.

For example, given the following intervals, if we sort by start time, choosing the first interval prevents us from adding another interval until after time 18. This blocks the remaining intervals from being added to the set of non-overlapping intervals, even though none of those intervals overlap with each other.

Bottom of Form

A screenshot of a computer

AI-generated content may be incorrect.

If instead we sort by end time, we can start by adding the intervals that end the earliest. Intuitively, this frees time for us to add more intervals as early as possible, and yields the correct answer.

A screenshot of a computer

AI-generated content may be incorrect.

**Non-Overlapping Intervals**

This is the basis for the question [**Non-Overlapping Intervals**](https://www.hellointerview.com/learn/code/intervals/non-overlapping-intervals), in which we are given a list of intervals and asked to find the maximum number of non-overlapping intervals.

We sort the intervals by their end times, and then iterate over each interval, keeping a count of all intervals that overlap with the last interval in the non-overlapping set. We return the total number of intervals minus the count of overlapping intervals.